VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

MONITOROVÁNÍ PEERŮ SDÍLEJÍCÍCH TORRENTY

TORRENT PEER MONITORING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. DAVID BEZDĚK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. LIBOR POLČÁK, Ph.D.**
SUPERVISOR

BRNO 2018

Zadání diplomové práce/20316/2017/xbezde11

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Bezděk David, Bc.**
Obor: Bezpečnost informačních technologií
Téma: **Monitorování peerů sdílejících torrenty**
Torrent Peer Monitoring
Kategorie: Počítačové sítě

Pokyny:

1. Seznamte se s protokolem BitTorrent.
2. Nastudujte metody pro detekci peerů sdílející konkrétní torrent.
3. Navrhněte metodu pro monitorování peerů sdílející konkrétní torrenty a návrh konzultujte s vedoucím práce.
4. Návrh implementujte.
5. Implementaci otestujte na vlastním trackeru i trackerech dostupných na Internetu.
6. Navrhněte možná rozšíření projektu.

Literatura:

- Stefan Schindler. Analysis of BitTorrent Trackers and Peers. Bakalářská práce, 2015 Friedrich-Alexander-Universität, Erlangen-Nürnberg.
- Anders Drachen, Kevin Bauer a Robert Veitch. Distribution of digital games via BitTorrent. In: Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments. ACM. 2011, str. 233-240.
- Chao Zhang et al. Unraveling the bittorrent ecosystem. In: IEEE Transactions on Parallel and Distributed Systems, 2011, str. 1164-1177.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Polčák Libor, Ing.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 23. května 2018

L.S.

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Tato práce se zabývá analýzou a realizací metod pro monitorování systému BitTorrent se zaměřením na protokol Mainline DHT. Cílem bylo vytvořit monitorovací systém, který bude vyhledávat uživatele BitTorrentu. Tito účastníci se nazývají peers a podílejí se na nelegální distribuci souborů. Dalším úkolem bylo získat a analyzovat data pro výpočet velikosti sítě. Toho bylo docíleno převzetím existující metody z odborného článku. Systém byl navržen a implementován tak, aby byl jedním z modulů pro monitorování kybernetické kriminality. Dále také definuje rozhraní pro ukládání i poskytování získaných dat, které slouží pro vyhodnocení a snadnou manipulaci s daty a tím umožňuje případná budoucí rozšíření.

Abstract

This master's thesis deals with analysis and implementation of methods for BitTorrent monitoring focusing on the Mainline DHT protocol. The aim of the thesis was to create a system, that will be looking for BitTorrent peers that participe in the illegal file distribution. Another task of the system was to collect and analyze data for counting size of the BitTorrent network. That was achieved by taking over of existing method. The system was designed and implemented as a module for monitoring of cybernetic crime. It also defines an interface for storing and sharing data, that provides data evaluation, easy data manipulation and serves for possible future extensions.

Klíčová slova

BitTorrent, torrent, tracker, peer, swarm, monitorování BitTorrentu, klient-server, peer-to-peer, P2P, BitTorrent protokol, distribuovaná hashovaná tabulka, DHT, Mainline DHT, MLDTH, btdht.

Keywords

BitTorrent, torrent, tracker, peer, swarm, BitTorrent monitoring, client-server, peer-to-peer, P2P, BitTorrent protocol, distributed hash table, DHT, Mainline DHT, MLDTH, btdht.

Citace

BEZDĚK, David. *Monitorování peerů sdílejících torrenty.* Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Libor Polčák, Ph.D.

Monitorování peerů sdílejících torrenty

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Libora Polčáka, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
David Bezděk
20. května 2018

Poděkování

Rád bych poděkoval vedoucímu této práce Ing. Liboru Polčákovi, Ph. D. za jeho vedení, trpělivost, poskytnutí celé řady užitečných rad a především za jeho neodkládané reakce a ochotu mi pomoci.

Obsah

Kapitola 1

Úvod

Počátkem 21. století se velmi rozmohl síťový systém zvaný BitTorrent, jehož autorem je americký programátor Bramem Cohanem. Jedná se o protokol spadající do architektury *Peer-to-peer síťových aplikací* (viz sekce 2.2), která se diametrálně liší od architektury *klient-server*. V principu je hlavním rozdílem těchto architektur přítomnost centrálního zařízení fungujícího jako server, což výrazně ovlivňuje jejich vlastnosti (viz kapitola 2).

Hlavním využitím BitTorrentu je peer-to-peer distribuce souborů, která je stále hojně využívaná. Jednou z nejznatelnějších výhod této distribuce je bezpochyby její rychlost nebo například schopnost vypořádat se s neustále rostoucí velikostí jednotlivých souborů. Výhodou je především absence serveru, což odstraňuje problém s přílišným zatížením centrálního serveru. To uvolňuje cestu ke komunikaci mezi jednotlivými koncovými zařízeními a tím pádem také k rovnoměrnějšímu rozložení zátěže a současně i zvýšení rychlosti. Tato komunikace má jeden hlavní problém, kterým je nelegální činnost v podobě neautorizované distribuce souborů. Proto je systém BitTorrent v hledáčku držitelů autorských práv, kteří se snaží tyto systémy monitorovat. Výsledkem monitorování BitTorrentu zmíněnými společnostmi vlastnící autorská práva bývá např. zasílání upozorňujících zpráv účastníkům BitTorrentu, že se podílejí na nelegální činnosti [12]. V některých případech je výsledkem jejich činnosti zablokování stránky poskytující služby BitTorrent. Mezi takto zablokované služby patří Kickass Torrents nebo třeba Mininova.

V posledním letech se však rozmáhají systémy s podobným účelem, které tvoří konkurenci a přebírají BitTorrentu miliony uživatelů. Jedná se o cloudově orientované systémy sloužící pro synchronizaci služeb na více zařízeních a zálohování dat jako jsou Dropbox, Microsoft Skydrive, Apple iCloud nebo např. Google Drive. Dalším konkurentem je placená služba americké společnosti Netflix, která se zaměřuje na online poskytování filmů a seriálů. Ta se těší velkému zájmu uživatelů, čímž přispívá k omezování nelegální činnosti probíhající v systému BitTorrent.

Předmětem této práce je monitorování koncových zařízení systému BitTorrent nazývaných *peers*. Každý *peer* je zařízení účastnící se distribuování konkrétního souboru nebo obsahu. Je tedy zapotřebí se zaměřit na tento obsah a určit pro něj, kteří klienti (*peers*) jej stahují, respektive distribuují dále. Podnět práce vzešel z potřeby projektu Tarzan, jehož hlavním smyslem je detekce a analýza kybernetické kriminality. Ta se může odehrávat v rámci různých prostředí jako jsou např. komunikační a mobilní aplikace nebo také Internet věcí (anglicky „*Internet of Things*“).

Postupně je zde rozebrána problematika síťových architektur s popisem jejich rozdílů (viz kapitola 2). Kapitola 3 se zaměřuje na samotný protokol BitTorrent. Konkrétně zde jsou rozvedeny informace o samotné funkčnosti systému, o proceduře přípojení nového peera

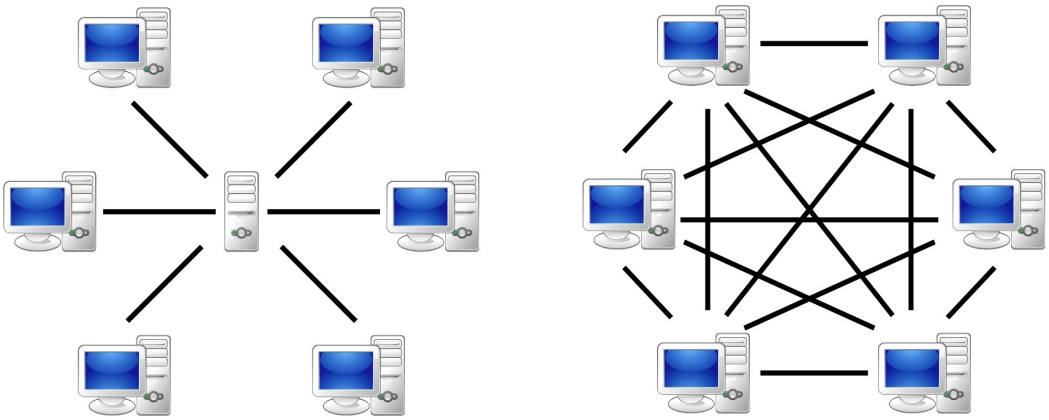
(viz sekce 3.1) a také o rozhodování peera (viz sekce 3.2). Následuje sekce 3.3 obsahující podrobnou analýzu samotného protokolu se zaměřením na protokoly jednotlivých entit systému jako je peer, tracker, metainfo soubor atd. Kapitola 4 rozebírá různé metody pro monitorování BitTorrentu se zaměřením na detekci peerů. Nejvíce je zde pak rozebrána metoda Mainline DHT, která je nerozšířenější metodou decentralizace systému BitTorrent pomocí distribuované hashované tabulky. Další kapitola 5 se zabývá analýzou požadavků a návrhem požadovaného systému pro monitorování. Implementace tohoto návrhu je popsána v kapitole 6. Závěrečná kapitola 7 se věnuje testování a vyhodnocení dosažených výsledků implementovaného systému.

Kapitola 2

Architektury síťových aplikací

Síťové aplikace jsou programy, které běží na odlišných zařízeních a komunikují s ostatními prostřednictvím sítě (typicky pomocí internetu). Architektura jednotlivých síťových aplikací bývá zpravidla navržena jejím designérem na základě povahy dané aplikace. Tradičně se dělí na architektury *klient-server* a *peer-to-peer*. Obecně lze říci, že základním rozdílem je především existence zařízení nazývané server, pro které jsou obvyklé určité rysy. Dalším charakteristickým rozdílem je vzájemná komunikace mezi účastníky, kdy u *klient-server* probíhá komunikace pouze mezi klientem a serverem. U *peer-to-peer* je pak typická komunikace mezi jednotlivými klienty [9].

Tato kapitola se zabývá podrobným popisem obou zmíněných architektur. Sekce 2.1 rozebírá *klient-server* a sekce 2.2 *peer-to-peer*. Závěrečná sekce 2.3 této kapitoly se věnuje podrobné analýze jejich rozdílu se zaměřením na porovnání rychlosti a náročnosti při distribuování souboru, což je funkčnost interesovaného systému BitTorrent.



Obrázek 2.1: Komunikace mezi zařízeními v architektuře klient-server (vlevo) a v architektuře peer-to-peer (vpravo)[1].

[1]Přebráno ze stránky: https://curlewresearch.com/lims-domain-ripe-4-disruption/

2.1 Architektura klient-server

Základním stavebním prvkem architektury klient-server je zařízení sloužící jako server, které bývá zpravidla stále aktivní a má za úkol obsluhovat požadavky od různých klientů. Typickým zástupcem jsou webové aplikace komunikující pomocí protokolu HTTP. Vyznačují se existencí aktivního webového serveru. Ten zpracovává a vyřizuje požadavky webových prohlížečů, které běží na klientských zařízeních. Komunikace zde vypadá tak, že webový prohlížeč žádá server o zaslání určitého objektu, který mu je poté serverem odeslán. Tím pádem zde neexistuje přímá komunikace mezi klienty, nýbrž se komunikuje pouze prostřednictvím serveru. Dalším charakteristickým rysem je statická IP adresa serveru z důvodu jeho snadného adresování.

Vedle již zmíněných webových služeb využívají tuto architekturu také služby jako např. DNS, FTP, e-mail atd.

2.2 Architektura Peer-to-peer

V architektuře *peer-to-peer* (P2P) je pouze minimální potřeba přítomnosti serveru. Koncová zařízení komunikují přímo mezi sebou bez jakýchkoliv zprostředkovatelů. Alternativní definice popisuje architekturu P2P jako systém, pro jehož zařízení jsou typické následující rysy: samoorganizovatelnost a decentralizované řízení [2]. Koncová zařízení nejsou ve vlastnictví žádného poskytovatele služeb, nýbrž patří jednotlivým uživatelům [9]. Jedním z nejznámějších zástupců této architektury je protokol BitTorrent, který slouží k distribuci souborů (viz kapitola 3). Dalšími reprezentanty jsou např. sítě Gnutella, KaZaA.

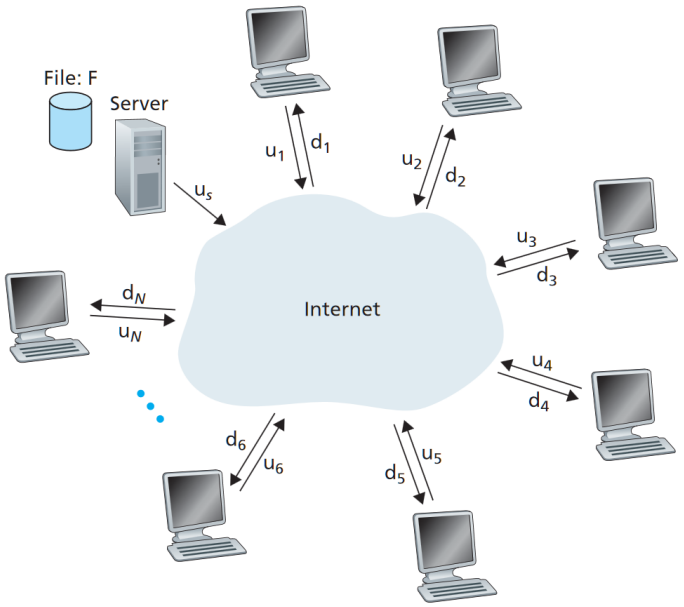
2.3 Porovnání architektur

Pro porovnání architektur klient-server a peer-to-peer lze využít příklad *distribuce souborů* [9]. Nechť na zařízení označeném jako server existuje soubor o velikosti f bitů (například nová verze operačního systému atp.), který je potřeba rozšířit mezi n uživatelů připojených k internetu (jak je vidět na obrázku 2.2). Rychlost nahrávání do sítě je označena u jednotlivých zařízení jako u_i. Rychlost stahování pak značí d_i. Výsledný čas je určen jako čas potřebný pro rozdistribuování souboru všem n zařízením.

V architektuře *klient-server* je nutné poslat kopii tohoto souboru n zařízením. Je potřeba tedy odeslat nF bitů. Pokud je rychlost uploadu serveru u_s, pak minimální doba pro distribuci do sítě D_{CSmin} bude lineárně závislá na počtu zařízení:

$$D_{CSmin} = \frac{nF}{u_s} \tag{2.1}$$

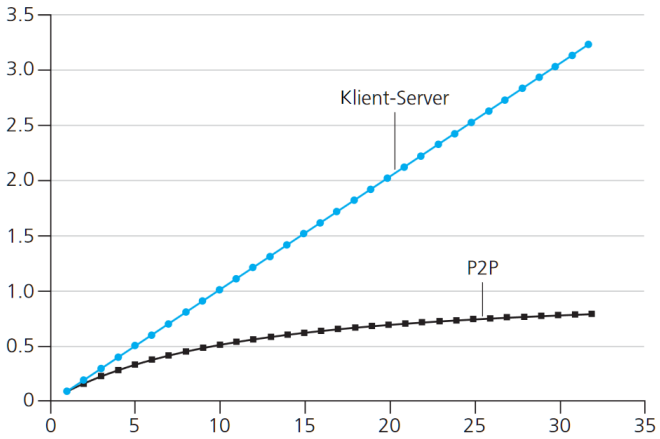
U architektury *peer-to-peer* je situace komplikovanější, ale zároveň efektivnější. Koncová zařízení, která již stáhla část souboru, mohou využít svou kapacitu pro upload a tuto část mohou dále distribuovat. Na počátku se soubor nachází opět pouze na jednom zařízení (peer), tento peer musí každý bit souboru odeslat alespoň jednou. Tedy minimální doba distribuce ze zdroje do sítě je F/u_s. Distribuce částí souboru pro všechny uživatele je však závislá na počtu cílových zařízení. I v tomto případě musí být rozdistribuováno nF bitů. To se rozloží mezi upload všech zařízení označený jako $u_{total} = u_s + u_1 + u_2 + ... + u_n$. Tedy výsledná minimální doba distribuce D_{PVPmin} je dána jako:



Obrázek 2.2: Ilustrace problému pro distribuci souboru (Převzato z [9]).

$$D_{P2Pmin} = \frac{nF}{u_s + \sum_{i=1}^{n} u_i} \tag{2.2}$$

Závěrečné porovnání minimální doby mezi n zařízení je zobrazeno na obrázku 2.3.



Obrázek 2.3: Zobrazení minimální doby (osa y) potřebného pro distribuci souboru v závislosti na počtu klientů (osa x) (Převzato z [9]).

Kapitola 3

BitTorrent

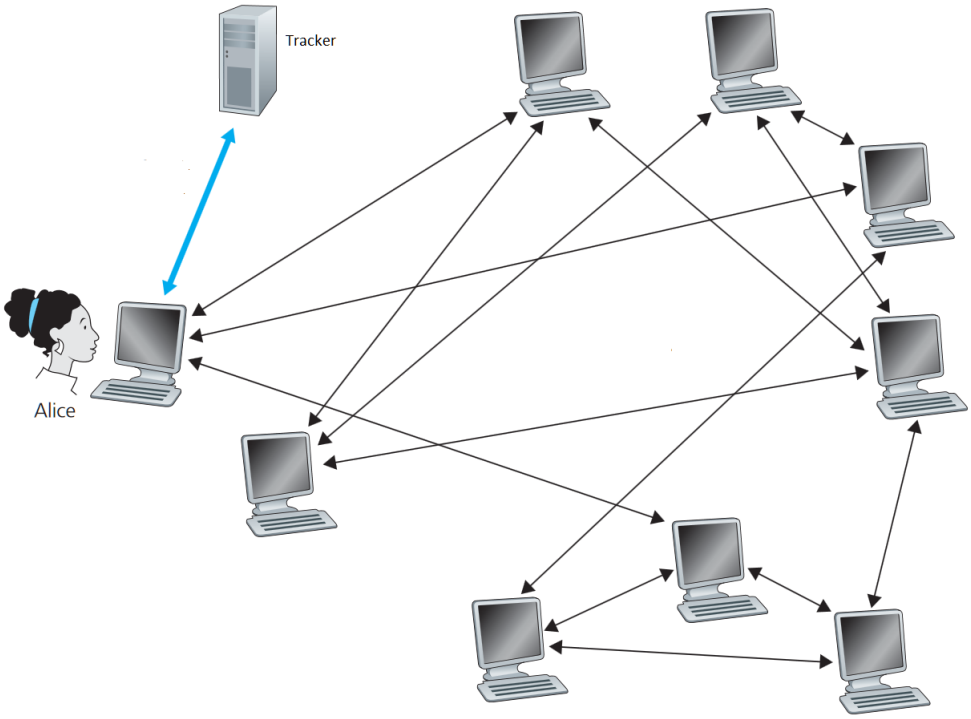
Protokol BitTorrent byl poprvé představen v roce 2001 a slouží pro distribuci objemných dat, jejíž příklad je uveden v sekci 2.3. Jeho úkolem je odstranění problémů při distribuci objemných souborů, především se snaží rozložit zátěž rovnoměrně mezi všechna zařízení. Nyní je třeba aplikovat terminologii tohoto protokolu na architekturu *peer-to-peer* [2, 6, 7, 9]:

- *peer* – Jak již bylo zmíněno v úvodu této práce, jedná se o koncové zařízení, které stahuje části jistého souboru zvané *chunks*. Tyto *chunks* mají stejnou velikost - typicky 256kB. V případě, že zařízení provádí stahování a zároveň i poskytuje tyto *chunks* dále, nazývá se *leecher*. Jestliže *peer* pouze poskytuje určité *chunks*, respektive již má nashromážděné všechny *chunks* konkrétního souboru, pak se nazývá *seeder*.
- *swarm* – Označení všech zařízení zapojených do distribuce určitého souboru.
- *tracker* – Jedná se o prvek, respektive službu systému, který zprostředkovává spojení v rámci *swarm*. Jeho úkolem je udržovat seznam *peers* zapojených do *swarm* a poskytovat jednotlivým *peers* náhodně vybranou podmnožinu těchto *peers*.

3.1 Připojení nového peera

Vedle samotné distribuce je však zapotřebí pochopit mechanismy připojení do *swarm* a následné rozhodování při výběru peerů, se kterými se bude komunikovat. Na obrázku 3.1 je mj. možno vysvětlit proces připojení. Nazvěme počítač, který ovládá uživatel Alice, jako nového peera. Poté zpravidla probíhají následující kroky:

- **1. krok**: Uživatel si nejdřívě na *tracker-discovery* stránce nalezne *torrent* soubor obsahující informace potřebné pro zahájení stahování, respektive pro nalezení a ohlášení se trackeru. Podrobnější rozbor náležitostí tohoto souboru je rozepsán v podsekci 3.3.2.
- **2. krok**: Nově připojený peer se ohlásí trackeru. Tracker poté náhodně vybere podmnožinu ze seznamu IP adres všech peerů a zasílá ji jako odpověď Alici.
- **3. krok**: Peer navazuje TCP spojení se všemi peery v zaslaném seznamu. Tyto peery můžeme nazvat jako sousedící.
- **4. krok**: V určitém čase má každý peer určitou podmnožinu chunků. Proto každý peer posílá periodicky sousedním uzlům dotazy na jejich seznam chunků. Podle získaných informací si poté peer žádá o zaslání chybějících chunků.



Obrázek 3.1: Příklad vzájemné komunikace uvnitř BitTorrent *swarm* (Převzato z [9]).

3.2 Výměna dat

V tento moment má Alice určité množství chunků a seznam, které chunky schraňují její sousedi. Na řadu přichází rozhodování. Je nutno rozhodnout:

1. O který chunk by mělo být zažádáno jako první?
2. Kterému sousedovi se zašle požadovaný chunk?

První problém je řešen technikou „*the rarest first*“, tedy z chunků, které právě Alice nemá, si vybírá ten, jehož kopií je nejméně mezi sousedy. Tím je zajištěno, že ty „nejvzácnější“ chunky se distribuují rychleji a je tak minimalizována pravděpodobnost jejich ztráty, která může nastat například při odpojení všech jejich držitelů od swarm.

Problém výběru souseda, kterému bude peer posílat svá data, je řešen následujícím principem. Každý peer si udržuje seznam několika málo sousedů, kteří mu poskytli nejvíce dat. Tento seznam je obnovován každých 10 sekund. Mimo to je každých 30 sekund náhodně vybrán jeden soused, kterému jsou zaslány chybějící chunky. Tímto je zvýšená pravděpodobnost, že se tato dvě zařízení vzájemně dostanou mezi nejfrekventovanější sousedy. Efekt je v tomto případě dvojí [5]:

1. Je pravděpodobné, že se mohou nalézt dvě zařízení, mezi kterými je rychlejší spojení.
2. Tato obměna peerů umožňuje zkontaktování nových účastníků, čímž se nový peer zapojí do komunikace uvnitř swarm a otevře se mu tak možnost účastnit se distribuce obsahu.

3.3 Protokol jednotlivých prvků

Vedle již zmíněných pojmů ohledně účastnících se entit, zmíněných v úvodu této kapitoly, patří mezi důležité pojmy také *tracker-discovery stránka* nebo *např. metainfo soubor*. Tyto dvě entity jsou spolu ve vztahu a to takovém, že webová stránka (tracker-discovery stránka), obsahuje velké množství metainfo souborů a tím poskytuje uživatelům přehled o dostupných souborech a zároveň tak umožňuje jejich sdílení. Podrobnější popis stránky je v podsekci 3.3.1 a metainfo souboru v podsekci 3.3.2.

3.3.1 Webová stránka

Jiným názvem *tracker-discovery* stránka. Jedná se o internetové stránky sloužící ke správě, uchovávání a poskytování metainfo neboli *torrent* souborů (viz podsekce 3.3.2). Některé stránky neposkytují metainfo soubory, ale nabízejí tzv. *magnet links*. Jejich účel je stejný. Hlavní rozdíl je v tom, že magnet link nepotřebuje být stažený, nýbrž může být rovnou zpracován BitTorrent klientem. Tento link je v podstatě kryptografický otisk obsahu daného souboru.

Webové stránky mohou být buď veřejné, které nevyžadují žádnou autorizaci a jsou volně dostupné, nebo privátní vyžadující přinejmenším registraci. Mezi světově nejznámější patří: Mininova.org (Nizozemsko), Thepiratebay.org (Švédsko), IsoHunt.com (Kanada) atd. [17]. Mezi české stránky patří např.: μTorrent[1], TrezzoR[2] atd.

3.3.2 Metainfo soubor

Metainfo soubor známý také jako **torrent** jsou bencodované slovníky obsahující podstatné informace pro připojení se ke swarmu sdílejícího konkrétní soubor. Je dostupný z webových stránek (podsekce 3.3.1). Všechny textové řetězce musejí být v tomto souboru zakodovány do UTF-8. Mezi podstatné informace, které v sobě slovník nese patří [6]:

- **announce** - Neboli ohlášení URL trackeru, který spravuje daný torrent.
- **info** - je slovník, který popisuje soubor(y) torrentu. Existují dvě varianty. První pro samostatný soubor bez adresářové struktury a druhou možností je více souborový torrent. Do této sekce patří také následující klíče:
  - *name* - Tento klíč ukazuje do zakódovaného UTF-8 řetězce, kde je doporučený název pro uložení souboru. Tento klíč je čistě informativní.
  - *pieces* - Odkazuje na řetězec, jehož délka je dělitelná číslem 20. Jinými slovy tento string obsahuje určitý počet stringů o délce 20 znaků, kde počet odpovídá počtu kousků (chunks). Každý tento string je SHA1 hashem daného chunku na konkrétním indexu.
  - *piece length* - Oznamuje počet bytů v každém kousku souboru (chunk). Zpravidla se jedná o mocninu čísla 2.
  - *length* - Tento klíč je ve vztahu s následujícím klíčem (*files*). Vyskytovat se může právě jeden z nich. Důvodem je to, že klíč *length* se používá u samostatných souborů a značí celkovou délku souboru v bytech.

[1] Dostupné na adrese: https://www.utorrent.cz/
[2] Dostupné na adrese: https://tracker.czech-server.com/prihlasenie.php

– *files* - Nachází se pouze u více souborového torrentu. Jedná se o seznam slovníků, kde pro každý soubor existuje jeden slovník. A v každém se nachází následující klíče: *length*, který značí délku daného souboru, a klíč *path*.
– *path* - Jedná se o jeden z klíčů ve slovnících u více souborových torrentů. Opět značí seznam stringů. Zde však dohromady reprezentují cestu a jméno souboru.

3.3.3 Tracker

Tracker je HTTP/HTTPS služba zpracovávající a odpovídající na požadavky GET. Tento GET request typicky obsahuje informace o klientovi. Mezi podstatné parametry patří [6]:

- *info_hash* - 20 bytový SHA1 hash benkodované hodnoty části *info* z metainfo souboru.
- *peer_id* - Řetězec o délce 20 znaků, který peer používá jako své ID. Tento řetězec si klient generuje sám při zahájení stahování.
- *port* - Číslo portu na kterém peer naslouchá. Typicky se používá port 6881, případně 6882 až 6889.

Odpověď od trackeru pak obsahuje bencodované slovníky. Pokud dojde k selhání, tracker odpoví s klíčem *failure reason*, následovaný lidsky čitelným stringem. Jinak musí obsahovat dva klíče:

- *interval* - Počet sekund, které má peer čekat mezi odesláním dvou regulérních zpráv.
- *peers* - Pod tímto klíčem je schován požadovaný seznam slovníků jednotlivých peerů, kde každý slovník obsahuje tyto klíče: *peer id, ip*, a *port*.

3.3.4 Peer

Protokol pro klienty je na obou stranách komunikace symetrický. Jedná se o komunikaci mezi dvěma peery, kde může docházet k toku dat oběma směry. Typicky zde jsou indexovány části souborů, které odpovídají těm v metainfo souboru. Jakmile má peer stažené všechny tyto části, oznámí to všem „svým“ peerům. Spojení musí obsahovat dva stavové bity: První je **choked**, který značí, zda je daný klient zahlcený, respektive zda nemůže odpovídat na dotazy, či není zahlcený a může reagovat. Druhý je **interested** a říká, zda je klient zainteresován něčím, co může druhý peer nabídnout, nebo zda se peer zajímá o něco, co může nabídnout klient. Přenos je tedy zahájen v případě, že jedna strana je *interested* a požadovaná strana je *unchoked*.

Jelikož klienti peerů spolu zpravidla komunikují pomocí TCP spojení (existuje i UDP [6]), spojení se zde iniciuje pomocí zpráv **Handshake**. Pro všechny zprávy po handshake platí, že datový typ integer je zakodován jako 4-bytová big-endian hodnota. Struktura zprávy handshake: <pstrlen><pstr><reserved><info_hash><peer_id>

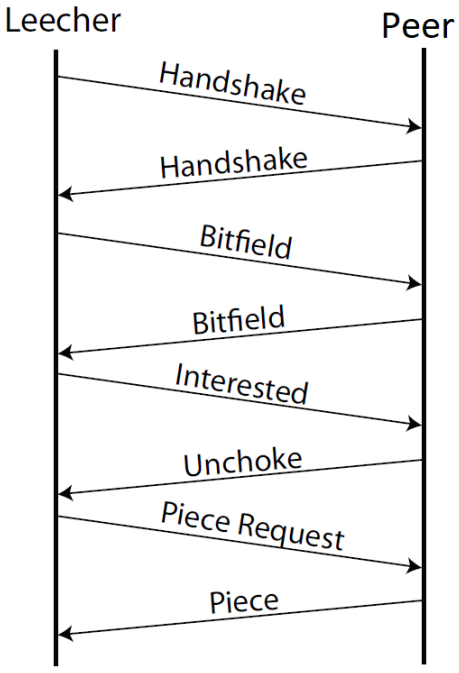
- *pstrlen* - Délka *pstr*
- *pstr* - Řetězec určující protokol.
- *reserved* - 8 rezervovaných bytů (samé nuly).
- *info_hash* - 20 bytový SHA1 hash bencodované hodnoty info z metainfo file (stejná hodnota jako info_hash v tracker requestu s tím rozdílem, že zde to není ve formě quoted).

- *peer_id* - peer id, které bylo posláno od trackeru a se kterým se snaží klient spojit.

Pokud obě strany nezašlou stejnou hodnotu *info_hash* nebo *peer_id*, dojde k přerušení spojení.

Všechny zprávy, kromě zpráv udržujících spojení a které jsou ignorovány, začínají jedním bytem, který určuje jejich typ. Existující následující možné hodnoty [6, 1]:

- 0 = **Choke**, 1 = **Unchoke**.
- 2 = **Interested**, 3 = **Not Interested**.
- 4 = **Have** - Číslo, určující index části, která byla úspěšně stažena a zkontrolována.
- 5 = **Bitfield** - Tato zpráva je poslána v případě, že je úspěšně ukončeno navázání spojení pomocí *Handshake*. Jedná se o nepovinnou zprávu, která říká, jaké části již daný klient má stažené.
- 6 = **Request** - Žádost určitou část souboru.
- 7 = **Piece** - Zaslání části souboru.
- 8 = **Cancel** - Zasláno po dokončení stahování.



Obrázek 3.2: Sekvence zpráv při navázání komunikace dvou peerů a následném odeslání části souboru (Převzato z [1]).

3.4 DHT Protokol

DHT protokol (*Distributed Hash Table*) čili distribuovaná hashovaná tabulka v systému BitTorrent bez trackerů slouží pro uložení seznamů peerů sdílejících určitý obsah. Jedná se o způsob decentralizace architektury založené na trackeru. Řešením je ukládání dvojic *key-value* (klíč-hodnota) implementované jako P2P síť, kde klíč je infohash obsahu a hodnota je seznam uzlů, což je zmíněná distribuovaná hashovaná tabulka. Protokol DHT vychází z Kademlia protokolu, který komunikuje přes UDP [10, 15].

Dalším rozdílem protokolu DHT je vedle komunikace přes UDP také jiná terminologie. Peer je klient nebo server poslouchající na určitém TCP portu a implementuje protokol BitTorrent. Uzel je klient nebo server poslouchající na UDP portu implementující DHT. Systém DHT je tvořen uzly, které mají uložené umístění peerů. Klient BitTorrentu obsahuje uzel, který je používán pro komunikaci s ostatními DHT uzly za účelem zjištění lokace ostatních peerů [10].

Každý uzel je identifikován *klíčem uzlu* (node ID). Ten má velikost 160 bitů (20B) a není trvalý, nýbrž se generuje v závislosti na implementaci protokolu DHT. Dále uzel mění své node ID v případě, že hledá soubor, a snaží se přiblížit své ID k ID obsahu. Stejnou velikost má pak ID obsahu, který se zde nazývá opět *infohash*. Uzly i obsah mohou mít několik různých identifikátorů v jednom jmenném prostoru a každý uzel je zodpovědný za udržování kopie obsahu, který je danému uzlu v prostoru blízko. Vzdálenost se počítá jak mezi uzly, tak mezi uzlem a obsahem. Uzel, který vykazuje menší vzdálenost k ID obsahu, jej bude s větší pravděpodobností znát. Vzdálenost uzlu A od uzlu B $d(A, B)$ se určí jako výpočet XOR hodnoty jejich klíčů:

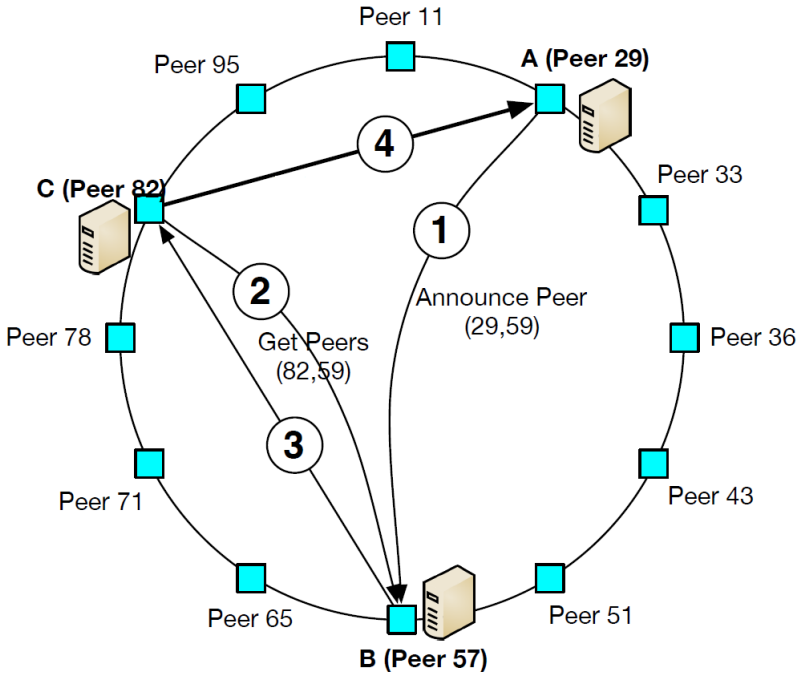
$$d(A, B) = ||A \oplus B|| \tag{3.1}$$

kde $||$ značí kardinalitu rozdílu jednotlivých bitů. Čím je vypočítané číslo menší, tím jsou dané uzly k sobě blíže. Stejným způsobem lze zjistit také vzdálenost ID uzlu a ID obsahu.

Distribuovaná hashovaná tabulka je v podstatě seznam uzlů, kterou si každý uzel udržuje. Také ji lze nazvat jako *routovací tabulku*. Typicky v ní uzel uchovává seznam aktivních uzlů. Zaznamenávání uzlů do této tabulky probíhá např. při hledání obsahu, kdy se do tabulky zapisují ty uzly, které mají malou vzdálenost k danému obsahu. Při dotazu jiného uzlu na obsah s určitým ID vrací uzel jako odpověď několik uzlů, u kterých je pravděpodobné, že budou daný obsah znát. Tím je nahrazena funkčnost trackeru.

DHT zpravidla podporuje čtyři typy zpráv [10, 14]:

- *ping* - Zpráva, která ověřuje dostupnost uzlu. V případě, že nedorazí odpověď, je daný uzel vyřazen z routovací tabulky.
- *find_node* - Parametrem této zprávy je cílové node ID (klíč uzlu) a jejím cílem je získání hledaného uzlu případně k nejbližších sousedů k danému uzlu. V případě, že uzel nezná dané node ID, vrací uzly z jeho routovací tabulky, u kterých je nejpravděpodobnější, že dané ID znají (mají nejmenší vzdálenost). Tím se daný uzel dostává blíže a blíže, dokud nenalezne hledaný uzel.
- *get_peers* - Je zadán infohash souboru a očekává se obdržení seznamu peerů. Princip fungování je stejný jako u find_node s rozdílem, že zde je místo node ID zadán infohash souboru.
- *announce_peer* - Ohlášení uzlu, že patří do swarmu.



Obrázek 3.3: Příklad komunikace pomocí DHT (Převzato z [14]).

Nyní předpokládejme BitTorrent swarm využívající DHT. Jak je vidět na obrázku 3.3, komunikaci lze ukázat na třech uzlech A (ID 29), B (ID 57), C (ID 82). A drží soubor, jehož infohash má hodnotu $x = 59$. B je zodpovědný za držení seznamu uzlů pro x, jelikož jeho node ID je blízké hodnotě x, a C chce stáhnout tento soubor [14].

1. Krok: Nejdříve A zveřejní soubor tím, že uloží x k uzlu B. Pro to však musí A iterativně volat *get_peers*, aby se dostalo k B. Následuje zpráva *announce_peer*, která oznámí uzlu B, že sdílí soubor s infohash x. B uloží kontakt na A do příslušného seznamu uzlů pro infohash x.

2. Krok: C chce nyní stáhnout soubor s infohash x. Postupuje podobně jako A, tedy posílá zprávy *get_peers* různým uzlům. Výsledkem je zaslání zprávy *get_peers* právě uzlu B.

3. Krok: Uzlu C je od B odeslán seznam uzlů. V tomto případě je v seznamu pouze uzel A. V tento moment začíná komunikovat uzel C s uzlem A pomocí klasického BitTorrent protokolu.

Existují různé implementace protokolu DHT. Mezi nejznámější patří Vuze a Mainline DHT (MLDHT). Obě tyto implementace jsou založeny na protokolu Kademlia. Zásadní rozdíl je v tom, že MLDHT seznamy peerů jsou nalinkovány na aktuální infohash souboru. Zatímco Vuze DHT mapuje peery na SHA-1 hashe původních infohashů. Dalším rozdílem je např. přidělování ID, které ve Vuze probíhá na základě IP adresy daného peeru. Oproti tomu MLDHT generuje při každém novém připojení do systému náhodné číslo, které je poté určeno jako ID. Další rozdíly jsou například odlišné názvy zpráv a další implementační detaily. Tato práce se bude soustředit především na Mainline DHT z důvodu jejího většího rozšíření [14, 16].

3.4.1 KRPC Protokol

KRPC protokol je protokol zabývající se formátem jednotlivých zpráv zasílaných v rámci protokolu DHT. Jedná se o zprávy popsané výše (*ping, find_node, get_peers, announce_peer*). Tento protokol říká, že jednotlivé zprávy jsou slovníky zakódované pomocí kódování bencode posílané jako UDP sockety [10].

Zpráva KRPC obsahuje slovník se třemi klíči, které jsou typické pro každou zprávu, a několika dodatečnými klíči. Klíče vypadají následovně:

- Klíč „t“ - Každá zpráva má klíč „t“ obsahující hodnotu reprezentující ID transakce. Toto ID je generováno dotazujícím se uzlem a slouží pro ověření, zda odpověď patří právě k dané zprávě.
- Klíč „y“ - Každá zpráva také obsahuje klíč „y“. Hodnota pod tímto klíčem obsahuje právě jeden znak určující typ zprávy. Zpráva může jedním z následujících typů: „q“ (query), „r“ (response) nebo „e“ (error). Do kategorie „q“ spadají i výše zmíněné zprávy *ping, find_node, get_peers, announce_peer*.
- Klíč „v“ - Posledním klíčem, který už není povinný, ale bývá často ve zprávách obsažen je klíč „v“, což jsou dva znaky identifikující klienta spolu se dvěma znaky definujícími jeho verzi.

Zprávy obsahující v klíči „y“ hodnotu „q“ jsou v kontextu této práce důležité, jelikož se jedná o dotazy, které budou monitorovací moduly odesílat. Jestliže je v klíči „y“ hodnota „q“, pak se ve slovníku nachází klíče:

- Klíč „q“ obsahující typ query zprávy ve formátu string (ping, find_node, get_peers atd.).
- Klíč „a“ slovník obsahující dodatečné informace podle příslušného typu query zprávy.

Zprávy, které mají pod klíčem „y“ hodnotu „r“ značící odpověď, obsahují dodatečný klíč „r“, pod kterým se skrývá slovník s požadovanými hodnotami. Jako odpověď na dotaz *find_node* obsahuje tento slovník hodnoty ohlášených uzlů pod klíčem „nodes“. Odpověď na dotaz *get_peers* pak typicky obsahuje klíč „nodes“, pokud daný uzel nezná hledaný soubor a nebo „values“ v případě, že zná daný soubor a může poskytnout peery. Pro protokol IPv6 je podstatným rozdílem pouze název klíče pro ohlašované uzly. Informace o uzlech nacházejí pod klíčem „nodes6“ [4].

Důležitým pojmem z tohoto protokolu je tzv. „*kompaktní formát*“ informace:

- Pro IPv4 - kompaktní formát je speciální jak pro uzel, tak i pro peera [10]:
  - Kompaktní formát peera je 6-bytový string, kde 4 byty jsou IP adresa a 2 byty jsou port.
  - Kompaktní formát uzlu je 26-bytový string, kde začátek tvoří 20 bytů značících ID uzlu, zbytek jako u peera.
- Pro IPv6 - formát pro IPv6 je rozdělen stejně jako u protokolu IPv4 [4]:
  - Kompaktní formát peera je 18-bytový string, kde 16 bytů tvoří IP adresu a 2 byty značí port.
  - Kompaktní formát uzlu je 38-bytový string, kde začátek tvoří 20 bytů značících ID uzlu, zbytek jako u peera.

Pro protokol IPv6 existují další drobné odlišnosti. První odlišností je to, že sockety by měly být zaslány přes protokol IPv6. Toto nemusí být vždy nutné, jelikož se lze dotazovat na IPv6 adresy i přes IPv4 sockety. V tomhle případě však musejí být dodržené určité podmínky. Především je nutné explicitně specifikovat požadavek na IPv6 uzly. Takový požadavek je vytvořen přidáním parametru *want* do zprávy. Ten může nabývat hodnot [4]:

- „n4“ - Tato hodnota značí, že uzel v dotazu požaduje uzly typu IPv4, a proto bude odpověď obsahovat parametr *nodes*.
- „n6“ - Hodnota n6 znamená, že se uzel poptává po IPv6 uzlech. To znamená, že odpověď zahrnuje i parametr *nodes6*, je-li to možné.
- Obou dvou hodnot n4 i n6.

U parametru *want* je důležitý poznatek, že hodnoty pod klíčem *values* nikdy nesmí obsahovat mix IPv4 a IPv6 adres. Také z pohledu specifikace protokolu není přijatelné, aby byly přes IPv4 posílány hodnoty *values* obsahující IPv6 peery a naopak. To znamená, že parametr *want* může ovlivnit pouze přítomnost hodnot *nodes* a *nodes6*.

Kapitola 4

Metody pro detekci peerů

Tato kapitola se věnuje různým přístupům k monitorování BitTorrentu, respektive jeho uživatelů podílejících se na sdílení souborů. Jejich cílem je identifikovat a shromažďovat IP adresy účastníků konkrétního torrent souboru nebo například ověřovat jejich aktivitu, případně validitu. Klient, který provádí takovou činnost se nazývá *monitorovací agent* [12]. Nejdříve zde jsou rozebrány *metody založené na dotazování trackeru*, které získávají požadované informace zasíláním požadavků na tracker spravující konkrétní obsah (viz sekce 4.1). Dále zde jsou popsány tzv. *Metody založené na DHT* (Distributed Hash Table), ve kterých dochází k získávání IP adres pomocí komunikace s jinými peery (viz sekce 4.2).

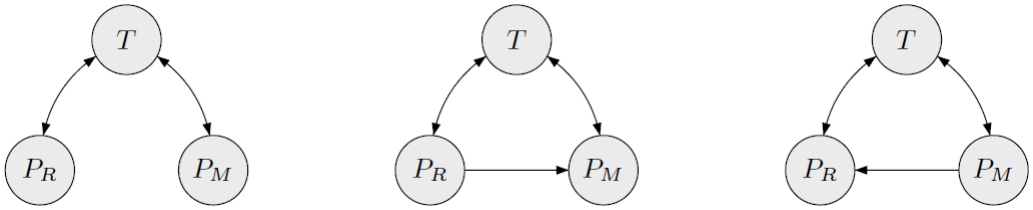
4.1 Metody založené na dotazování trackeru

Základním stavebním prvkem metod založených na dotazování trackeru (nebo také *Tracker-based* metody) je dotazování se centrálních trackerů na podmnožinu peerů v daném torrentu. Obecně lze tyto metody rozdělit dvěma způsoby:

1. Podle toho s kým komunikuje monitorovací agent:
   - Nepřímé - Monitorovací agent komunikuje pouze s trackerem, nikoliv s ostatními peery.
   - Přímé - Monitorovací agent komunikuje především s ostatními peery, ale může i s trackerem.
2. Zda monitorovací agent komunikuje s ostatními peery.
   - Aktivní - Úkolem těchto metod je aktivně získávat informace ať už posíláním dotazů na tracker nebo jednotlivým peerům.
   - Pasivní - Činnost agentů spočívá v pasivním přijímání zpráv od ostatních peerů.

Přímé metody zpravidla značí i aktivní a jsou oproti nepřímým doplněny o tzv. *probe zprávy*, které mají za úkol kontaktovat všechny peery a zjistit, zda dané peery opravdu existují a hlavně, zda se opravdu podílejí na sdílení souborů. Jedná se tedy o větší zapojení agentů a tím pádem zde jsou i větší požadavky na zdroje než u nepřímých metod. Dosažené výsledky však jsou daleko spolehlivější [1, 12, 3].

Na tyto metody proběhla celá řada studií, mezi kterými vyčnívá Piatek et al. [12], která se zaměřuje na identifikaci a vyhodnocení efektivnosti jednotlivých metod. Především je zde zaveden pojem **false positives**. K tomuto jevu dojde v případě, že je určitý peer



Obrázek 4.1: Ukázka možných tracker-based způsobů monitorování BitTorrentu v závislosti na směru komunikace. T značí tracker, P_R regulérní peer a P_M značí monitorovací peer (agent). Zleva: nepřímé monitorování, nepřímé a pasivní monitorování, přímé a aktivní monitorování (Převzato z [3]).

v libovolném swarm určen jako peer podílející se na nelegální distribuci, a to i v případě, že se ničeho takového nikdy nedopustil. Výsledky zde jsou prezentovány na vzorku zařízení, mezi kterými byla různá zařízení jako počítače, tiskárny, routery atd. připojená k síti. Přestože se některá zařízení nemohou podílet na sdílení souboru, obdrželo velké množství z nich upozornění, že se podílejí na nelegální aktivitě (v USA tzv. DMCA complaints).

Důvodem těchto false positives je typicky některá ze slabin BitTorrentu jako jsou např. možnost podvrhnutí IP adresy trackeru, chybná identifikace způsobená přihlašováním uživatelů z veřejných sítí (restaurace, kavárna) atd. u pasivních metod. Nebo například možnost provedení Man-in-the-Middle útoku jak u pasivních tak i u aktivních metod [12]. Mimo útoky na slabiny je také další možnou příčinou těchto false positives vkládání libovolných, ale validních IP adres do peer listů jednotlivých trackerů. Tato aktivita probíhá např. v jednom z největších a stále aktivních BitTorrent serverů - Pirate Bay [1].

Monitorovací agenti využívající tyto metody (ať už nepřímé či aktivní) vykazují typické rysy chování, čímž se stávají relativně snadno detekovatelnými. Znalostí těchto charakteristických rysů chování lze vytvořit monitor tak, aby byla co možná nejvíce eliminována pravděpodobnost jeho odhalení. Mezi hlavní rysy nepřímých metod patří [3]:

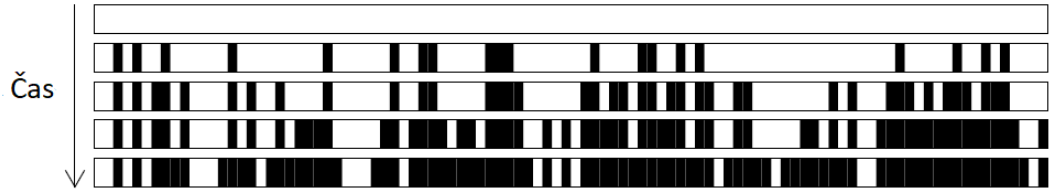
- 1. Monitorovací agentury využívají k monitorování jednu podsíť.
- 2. Čas, který agent stráví ve swarm, bývá typicky větší než u běžných peerů.
- 3. Počet různých kombinací (IP, port, infohash) od jedné IP adresy, respektive více klientů na jedné IP adrese.
- 4. Agenti typicky blokují všechny příchozí pokusy o spojení.
- 5. Počet swarmů, ve kterých se objeví IP adresy z jedné podsítě. Agentury typicky ze své podsítě monitorují větší množství swarm.
- 6. Počet, kolikrát se stejný (IP, port) pár souběžně objeví ve více swarmech.

U aktivních metod se vyskytují typické rysy chování. Jedná se o dva konkrétní rysy [3]:

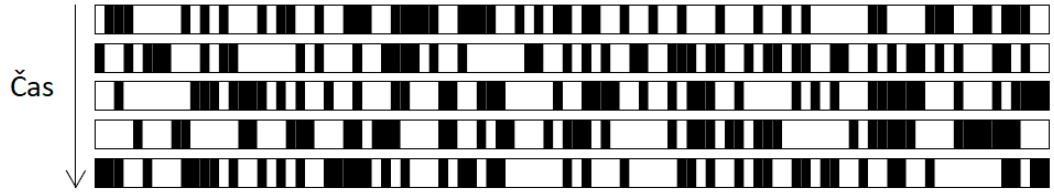
- 1. *Ohlášené dokončení* - Součástí zpráv *bitfield* je informace o tom, jaké bloky souboru má již daný peer stažený. U některých klientů se zdála tato informace jako nahodilá v určitém procentuálním intervalu. Respektive se procento jimi vlastněných bloků nezvyšovalo a navíc peer nahlásil blok na určitém indexu jako zatím nestažený, přestože jej krátce před tím potvrzoval jako stažený (viz obrázky 4.2 a 4.3).

- 2. *Frekvence připojení* - Je obvyklé, že se peer po nějaké době znovu přihlásí k určitému peeru v daném swarm. Monitorovací agenti se však ke stejnému peeru přihlásili po zhruba třikrát delší době než bylo v daném swarmu běžné pro regulérní peery [3].

Tracker-based metody tedy nejsou tak spolehlivé a přesvědčivé, jak by se na první pohled mohlo zdát. Prvním z důvodů je výskyt *false positives*, dalším pak například relativní jednoduchost detekce monitorovacích agentů. Ze zmíněných metod vykazují nejhorší výsledky ty nepřímé. Lepší výsledky již vykazovaly aktivní varianty, které se vyznačují především ověřováním a zpřesňováním výsledků pasivních metod [1, 12].



Obrázek 4.2: Progres stahování u regulérního peera. Černě jsou označeny části souboru označené jako stažené (Převzato z [3]).



Obrázek 4.3: Progres stahování u monitorovacího peeru (Převzato z [3]).

4.2 Metody založené na DHT

Metody založené na Distributed Hash Table neboli *DHT-based* metody slouží k monitorování BitTorrentu bez trackerů. Tento způsob monitorování se nazývá *DHT crawling*, volně přeloženo jako *prolézání*. Jeho principem je shromažďování infohash jednotlivých souborů, na jejichž základě se snaží kontaktovat jednotlivé uzly. Zasíláním zpráv get_peers nebo find_node monitorovací agent postupně získává uzly pro další hledání nebo konkrétní peery sdílející daný obsah (podrobnější popis viz sekce 3.4). Tato práce se věnuje rozšířenější variantě DHT, kterou je Mainline DHT.

4.2.1 Monitorovací agent

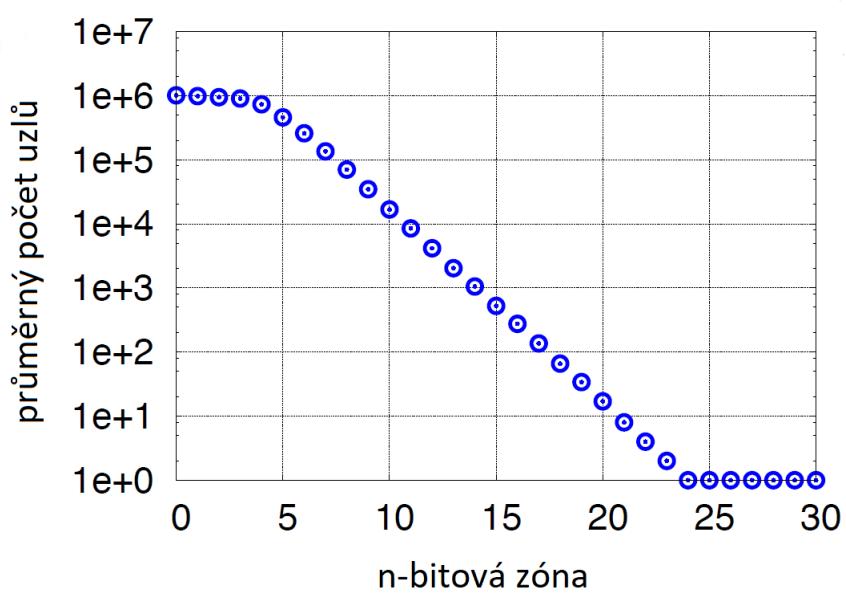
První metoda je implementace monitorovacího agenta chovajícího se podobně jako běžný uživatel. To znamená, že agent na základě známého infohashe hledá peery sdílející daný soubor [15]. V takovém případě agent postupuje jako ve 2. a 3. kroku příkladu komunikace pomocí DHT (viz sekce 3.4).

4.2.2 Zónový agent

V souvislosti s DHT se vyskytují tzv. *n-bitové zóny*, ve kterých uzly sdílejí n-bitový prefix v jejich node ID. Jelikož je stavový prostor určený těmito prefixy příliš velký a jeho prohledávání by trvalo příliš dlouho, je zapotřebí se zaměřit pouze na určitou podmnožinu prostoru prefixů. Graf 4.4 ukazuje přibližné rozložení uzlů v jednotlivých zónách. Na základě tohoto rozložení je vhodné určit rozsah prefixů, na který se daná metoda zaměří. Důvodem jsou nevýhody jako je přilíš velké množství uzlů v zóně (malé n) a tedy i velmi dlouhá doba prohledávání. Pro příliš malé zóny (větší n) je zase naopak problém s chybějícími uzly, což může znamenat chybějící informace o dalších částech zóny [14].

Prohledávání prostoru konkrétní n-bitové zóny slouží mj. pro přibližný odhad velikosti sítě nebo n-bitové zóny. Skládá se z následujících kroků [14]:

1. Nastavení node ID zónového agenta na náhodnou hodnotu.
2. Prohledávání BitTorrent systému pomocí zpráv find_node.
3. Hledání uzlů, které jsou dostatečně blízké nastavené node ID.
4. Slučování získaných výsledků od různých agentů.



Obrázek 4.4: Naměřený počet uzlů v jednotlivých n-bitových zónách v roce 2013 (Převzato z [14]).

Tuto metodu provází problém chybějících uzlů, který způsobuje značné zkreslení výsledků ohledně velikosti sítě. Příčin může být mnoho od např. nastavení firewallu, síťového připojení, abnormálnímu chování peerů atd. Pro omezení této chyby lze využít tzv. Bernoulliho proces. Jedná se o pravděpodobnostní distribuci s dvěmi třídami jevů. Buď může být uzel nalezen, nebo se jedná o uzel chybějí (nenalezený). Pokud je pravděpodobnost toho, že bude uzel nalezen je p, pak pravděpodobnost toho, že bude daný uzel chybějící je $1 - p$. Čili pro přibližné zjištění velikosti sítě je zapotřebí získat počet uzlů, ze kterého se určí hodnota p a pak lze snadno dopočítat množství chybějících uzlů [14].

Kapitola 5

Návrh monitorovacího systému

Návrh systému je založen na specifikaci požadavků, která je rozebrána v sekci 5.1. Systém lze rozložit do tří samostatných celků. První částí systému jsou jednotlivé moduly pro monitorování BitTorrentu. V tomto návrhu jsou zahrnuty dvě konkrétní metody pro monitorování, které mohou být obohaceny o další moduly (viz sekce 5.2). Druhou částí systému je databáze a aplikační rozhraní, které umožňuje práci s touto databází. Posledním samostatným celkem je zpracování dat, které bude z databáze číst získaná data. Poté bude následovat zpracování a vyhodnocení dat nebo např. porovnání výsledků (viz sekce 5.3). Tyto části jsou znázorněny na diagramu 5.1. Samotný běh systému bude řízen pomocí skriptu, který bude jednotlivé moduly spouštět a případně i ošetřovat neočekávané stavy atd.

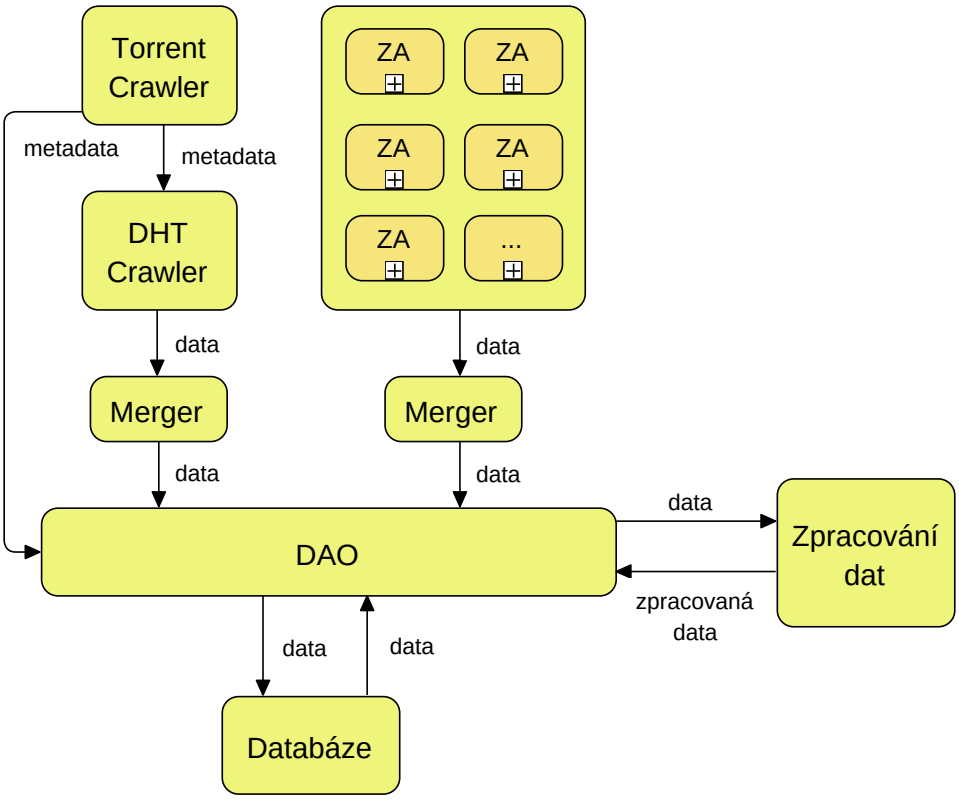
5.1 Analýza požadavků

Cílem systému je poskytnout informace o monitorované síti BitTorrent. Tyto informace by poté měly být dále použitelné jinými aplikacemi či projekty. Mezi základní požadavky tedy patří, aby daný program:

- Definoval a implementoval rozhraní pro poskytování dat (případně i pro přijímání parametrů jako např. URL webových stránek poskytujících seznamy torrent soborů či magnet linků atp.).
- Byl snadno upravitelný i rozšiřitelný.

Další důležitou součástí jsou pak funkční požadavky zaměřující se na jádro problému:

- Stažení a parsování souborů obsahujících torrent soubory.
- Rozšiřitelné rozhraní pro práci s databází.
- Zpracování metadat a jejich využití pro připojení do swarm a k následnému získání seznamu peerů.
- Zpracování získaných dat.
- Zobrazení případně jiné poskytnutí výsledků.



Obrázek 5.1: Dataflow diagram systému. ZA značí zkratku zónový agent. DAO značí Data Access Object tzn. objekty pro přístup k databázi.

5.2 Monitorování

Monitorování BitTorrentu je hlavním požadavkem na systém. Cílem je sbírat potřebná data pro jejich pozdější zpracování. Součástí této práce jsou dvě různé metody popsané v sekci 4.2. Na závěr každého monitorovacího modulu jsou data shromažďovány tzv. *mergerem*, který má za úkol odstraňovat redundantní data.

1. ***Monitorovací agenti*** - Jedná se o implementaci metody z podsekce 4.2.1. V tomto modulu se nacházejí dva podstatné prvky:

   - *Torrent Crawler* - Tento prvek bude volán jako první *Torrent Crawler*, jelikož jeho primárním cílem je obstarávání podstatných informací, respektive metadat ze serverů. Jeho druhým neméně podstatným úkolem bude zpracování těchto metadat a poté i jejich uložení do databáze.

     Před získáním metadat je však zapotřebí stáhnout z webových stránek tzv. RSS soubor. Typově se jedná o rodinu jazyku XML, což činí potřebné informace snadno dostupnými a také zpracovatelnými např. pomocí veřejně dostupných knihoven pro práci s XML formátem. Tento RSS obsahuje seznam sdílených souborů, kde ke každému z nich jsou přiloženy informace pro stažení torrentu (metadat).

     Na základě funkčnosti bude tento modul probíhat v cyklech, kde počet iterací bude určen počtem uvedených souborů. Počet souborů bude zjištěn z velikosti explicitně uvedeného seznamu URL. Každá iterace vypadá následovně:

     (a) Výběr URL požadovaného RSS souboru.
     (b) Stažení souboru se seznamem torrent souborů.
     (c) Parsování souboru.
     (d) Stažení torrent souborů.
     (e) Uložení příslušných informací do databáze a jejich přeposlání DHT Crawleru.

   - *DHT Crawler* - Tento prvek obstarává jádro řešení. Bude spuštěn po obstarání a zpracování metadat Torrent Crawlerem. Následovat bude připojení agenta do swarm a postupné hledání peerů sdílejících daný obsah. Z metadat je zjištěna hodnota infohash interesovaného souboru. Infohash je poté použit jako parametr u odesílaných zpráv get_peers. Tato zpráva je zasílána tak dlouho, dokud agent není dostatečně blízko, aby se mu podařilo získat seznam peerů.

2. ***Zónoví agenti*** - Modul se zónovými agenty je založen na procházení stavového prostoru peerů se zaměřením na konkrétní n-bitovou zónu. Modul obsahuje větší množství agentů, jejichž výsledky se poté budou slučovat opět pomocí prvku merger. Podrobný popis činnosti jednotlivých agentů je již popsán v podsekci 4.2.2. Každý agent tedy bude provádět následující kroky:

   (a) Nastavení node ID uzlu na náhodnou hodnotu.
   (b) Prohledávání stavového prostoru pomocí zpráv find_node.
   (c) Hledání uzlů spadajících do zvolené zóny.
   (d) Výpis, případně uložení, získaných výsledků.

   Návrh těchto agentů vychází z práce: „*Measuring Large-Scale Distributed Systems: Case of BitTorrent Mainline DHT*“ od autorů L. Wanga a J. Kangasharju [14].

5.3 Zpracování dat

Modul sloužící pro zpracování dat bude mít několik úkolů. Především se bude jednat o manipulaci se získanými daty, nad kterými se budou provádět vyhodnocovací, případně i jiné činnosti, mezi které patří:

- Slučování informací od podobných souborů. Tato činnost bude prováděna v případě, že bude detekováno více různých torrent souborů sdílejících např. stejný film s různými titulkami.
- Výpočet Bernoulliho procesu. Tento proces doprovází monitorování pomocí zónových agentů a slouží pro určení přibližné pravděpodobnosti, zda uzel bude nalezený nebo chybějící z pohledu agentů. Tímto způsobem lze také určit úspěšnost měření.
- Vyhodnocení získaných dat, případně zprůměrování různých měření.
- Porovnání vyhodnocených dat od různých modulů.
- Lokalizace uzlů / peerů podle IP adresy.
- Vizualizace výsledků.

Dále bude tato část systému sloužit jako prostor pro další případná rozšíření.

5.4 Aplikační rozhraní

Část systému v návrhu označená jako DAO bude mít funkci aplikačního rozhraní (API). Cílem této části systému je umožnění propojení více monitorovacích modulů. Toho bude docíleno pomocí vytvoření aplikace s klasickou architekturou klient-server, kde klient bude iniciovat komunikaci a bude určovat zda se budou zasílat data od klienta serveru nebo opačným směrem. Hlavními požadavky na tento program sloužící jako klient bude:

- Navázání spojení se serverem.
- Odeslání souborů obsahující data.
- Příjem souborů s daty.

Druhou částí tohoto systému tedy bude program server, který bude provádět hlavní myšlenku tohoto rozhraní. Server tedy bude:

- Pasivně přijímat požadavky od klienta.
- Přijímat soubory s daty.
- Ukládat poskytnutá data do databáze.
- Číst data z databáze na vyžádání od klienta.
- Odesílat soubory s požadovanými daty.

Kapitola 6

Implementace navrženého systému

Kapitola implementace obsahuje podrobný popis jednotlivých částí implementovaného systému. Všechny komponenty, které tvoří výsledný systém, lze vidět na obrázku 6.1. Vedle implementovaných částí tento obrázek obsahuje modul *BP*. Jedná se o externí modul, jehož integraci umožnila modularita a nezávislost jednotlivých komponent systému, díky čemuž by mohly být přidány i další práce zabývající se monitorováním systému BitTorrent jako samostatné moduly.

Významnou částí implementace je abstraktní třída nazvaná jako *AbstractCrawler*, která obsahuje jádro implementace pro DHT crawler (viz 6.2.2) i pro zónové agenty (viz 6.3.2). Jejím cílem je redukce duplikovaného kódu, respektive metod, které jsou využity v obou zmíněných modulech. Nejprve zde jsou deklarovány abstraktní metody jako např. *start_sender(), info()* atd. Následují implementace metod, jejichž kód je nezávislý na modulu, ze kterého jsou volány. Mezi implementované metody v abstraktní třídě patří např. *find_routers(), ping(), start_listener()* a další.

Kapitola implementace začíná sekcí 6.1 s popisem použitých technologií. Dále jsou rozebrány moduly pro monitorování. V této práci jsou moduly implementující dva různé přístupy, kde prvním je přístup hledající peery podílející se na sdílení určitého souboru (viz sekce 6.2) a druhý pak má za úkol prohledání sítě BitTorrent za účelem zisku co největšího množství uzlů, které jsou aktuálně připojeny do sítě (viz sekce 6.3). Následuje podkapitola věnující se pomocným zdrojovým souborům (viz sekce 6.4). Poté je v podkapitole 6.5 popsán server, na kterém běží rozhraní pro přístup k databázi a na kterém je i databáze samotná. Závěrečná sekce se věnuje modulům pro zpracování dat (viz sekce 6.6).

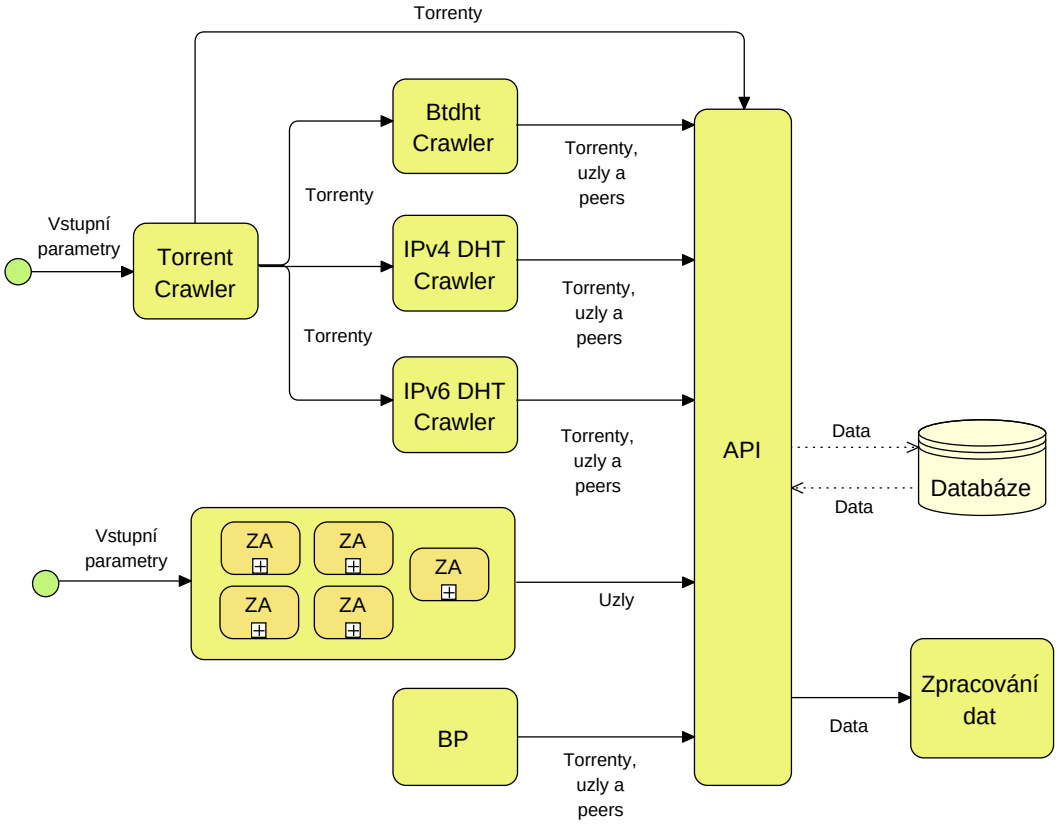
6.1 Technologie

Podkapitola technologie se věnuje využitým technologií v rámci implementace.

6.1.1 Python

Python je interpretovaný programovací jazyk, který nachází široké využití. Důvodem je jeho univerzálnost a skutečnost, že kombinuje procedurální, funkční a objektově orientovaná programovací paradigmata. Mezi jeho další přednosti patří snadná udržovatelnost a v neposlední řadě také jeho snadné využití na všech počítačových platformách, respektive jeho nezávislost na operačním systému. Jazyk Python je využíván celosvětovými firmami jako jsou Google, YouTube, Dropbox a další [11].

K volbě jazyku Python vedlo kromě zmíněných výhod také následující důvody:



Obrázek 6.1: Diagram obsahující komponenty implementovaného systému. Komponenta *BP* je bakalářká práce *Monitorovanie peerov BitTorrent na základe informácí z distribuovanej hašovacej tabuľky* od pana M. Vaška [13].

- Část práce zaměřená na prohledávání n-bitových zón vychází ze zdrojových souborů napsaných v tomto jazyce.
- Pro Python existuje knihovna, která implementuje funkci BitTorrent klienta a je využita i v této práci pro vyhledávání peerů podílejících se na sdílení souborů (viz podsekce 6.2.1) .

Převzaté zdrojové soubory byly napsány ve verzi jazyka 2.7, a proto je i pro ostatní moduly zvolena verze 2.7.

6.1.2 SQLite

SQLite je knihovna napsaná v jazyce C, která poskytuje odlehčenou diskově založenou databázi, která nevyžaduje běžící server. Jedná se o relační databázi, kde se pro přístup používá jazyk SQL [8]. Data se ukládají do souboru nebo do fyzické paměti. V případě ukládání souboru je celý datový model uložen do samostatného přenositelného souboru, který uchovává informace o architektuře databáze včetně uložených dat. Výhodou je vedle absence běžícího serveru také snadná přenositelnost na jiné platformy.

V jazyce Python je rozhraní pro knihovnu SQLite implementováno v modulu s názvem *sqlite3*, jehož autorem je Gerhard Häring. Modul poskytuje SQL rozhraní, které je v souladu se specifikací databázové API v jazyce Python [8].

6.2 Modul DHT Crawler

Modul pro monitorování sítě BitTorrent se zaměřením na získávání peerů, kteří se podílejí na sdílení konkrétních souborů se nazývá *DHT crawler*. Součástí implementace jsou tři varianty crawleru (viz obrázek 6.1). První využívá knihovnu s názvem *btdht*, a proto je zde nazvána jako: *btdht crawler*. Druhou a třetí variantou jsou crawlery IPv4 a IPv6, které se liší pouze v implementačních detailech. Oba lze označit jako DHT crawler, jejichž jádro tvoří třída DhtCrawler, a proto jsou tyto dva crawlery popsány v podkapitole *DhtCrawler*.

6.2.1 btdht crawler

Knihovna btdht je kompletní implementací BitTorrent Mainline DHT protokolu. Využití této knihovny je velmi jednoduché a rychlé. Nejprve je zavolána třída TorrentCrawler pro získání seznamu souborů, pro které se budou hledat peers. V souvislosti s monitorováním se z implementačního pohledu jedná pouze o zavolání knihovny, které se ponechá určitý čas pro bootstrapping. Bootstrapping je pasáž, která má za úkol získat co největší množství uzlů pro následnou lepší výchozí pozici při vyhledávání peerů. Následuje pouze zavolání metody *get_peers* a informativní výpis.

Navzdory své jednoduchosti má knihovna btdht i své nevýhody. Nevýhodou může být například méně informací, konkrétně je výstupem pouze získaný seznam peerů pro daný soubor, nelze však zjistit, který uzel informace poskytl. Podstatnějším nedostatkem však je to, že implementace existuje pouze pro protokol IPv4. Přestože by změna pravděpodobně byla možná, bylo by zapotřebí tyto úpravy vzhledem k robustnosti knihovny důkladně otestovat, a proto btdht crawler poskytuje pouze informace pro IPv4 protokol. Nicméně rozšíření knihovny pro protokol IPv6 by mohlo být jedním z možných rozšíření této práce.

6.2.2 DhtCrawler

Třída *DhtCrawler*, která dědí z abstraktní třídy *AbstractCrawler* a jejíž zakomponování v rámci hierarchie tříd tohoto monitorovacího systému, lze vidět v třídním diagramu na obrázku 6.3. Před samotným monitorováním se vytváří instance třídy ParamParser pro získání počátečního nastavení a třídy TorrentCrawler pro získání a případnou redukci či selekci ze seznamu souborů. Vedle parametrů pro získání RSS souboru lze nastavit i typ protokolu (IPv4 / IPv6) nebo např. parametr pro vypnutí výpisů. Hlavní výhodou tohoto programu oproti knihovně zmíněné v podsekci 6.2.1 je ona podpora IPv6 protokolu, která umožňuje prohledávání podstatně rozmanitějšího stavového prostoru. Navíc oproti btdht crawleru poskytuje třída DhtCrawler informace o uzlu případně o uzlech, které dané peery poskytly.

V principu lze monitorovací činnost rozdělit do následujících kroků:

- Střádání uzlů: Krok střádání uzlů lze anglicky nazvat jako *Bootstrapping*. Oproti zónovým agentům je toto střádání doplněno dodatečným zasíláním dotazů *find_node*. Vedle dotazování na nové uzly se posílají i dotazy, ve kterých je místo ID cílového uzlu vložen infohash souboru. Tím je zvýšena šance objevení uzlů, které mají menší vzdálenost k jednotlivým souborům a tedy i větší pravděpodobnost nalezení peerů. Tato fáze může být ukončena více způsoby, a to buď nedostatkem uzlů ve frontě, nebo dosažením určité hranice počtu uzlů.

- Hledání: V kroku hledání je cílem projít seznam vyhledávaných souborů a pro infohash každého souboru se snažit získat peery, které se podílejí na jeho sdílení. V aktuálním

seznamu uzlů se vyhledají ty s nejmenší vzdáleností vzhledem k hodnotě infohash. Následně jim je zaslána zpráva *get_peers*. Dle specifikace pak, buď uzly znají daný soubor a jejich odpověď obsahuje položku „values“ obsahující kompaktní formát peerů, nebo neznají daný soubor a zasílá v odpovědi pouze uzly ze své routovací tabulky, které jsou danému souboru nejblíže. Na konci vyhledávání peerů pro každý soubor provádí třída spojení (merge) ohlášených peerů do jednoho seznamu.

- Filtrování: Krok Filtrování má za úkol odstranění neexistujících uzlů. Filtrování probíhá ve dvou fázích, kdy v prvním kroku dochází k odstranění peerů s portem 1. Ve druhé fázi se pak prochází seznam všech peerů, kde každému peeru je zaslána zpráva ping. Filtrování pomocí pingu však vykazovalo velkou redukci výsledků, a proto je tato část filtrování ponechána jen pro účely vyhodnocení (viz podkapitola 7.1.1). K ukládání peerů dochází ještě před provedením této druhé fáze filtrování.

Na závěr dochází u ložení dat. Konkrétně se ukládají získané uzly do souboru: „ipv[4/6]nodes.datum_a_cas.id_uzlu.json“ a torrenty s nalezenými peery do souboru: „ipv[4/6]peers.datum_a_cas.id_uzlu.json“. Formát těchto souborů je popsán v podsekci 6.5.2. Vedle uložení dat je také vypsán stav nalezených uzlů, peerů, doba běhu atd.

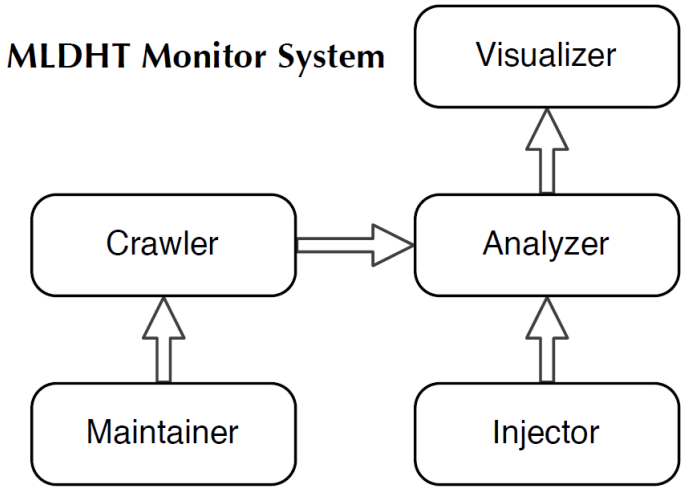
6.3 Zónový agent

Sekce věnující se zónovým agentům vychází ze zdrojových souborů převzatých z práce L. Wanga a J. Kangasharju [14]. Došlo však k úpravě souborů a to tak, aby bylo možné zasazení zdrojových souborů do třídního modelu (viz obrázek 6.3). Tyto zdrojové soubory vycházejí z architektury monitorovacího systému [14]. Architektura obsahuje dohromady pět logických celků. Jedná se o celky *maintainer, crawler, injector, analyzer, visualizer*.

Vedle samotného agenta (modul crawler) byly převzaty také zdrojové soubory implementující tzv. maintainer, injektované uzly a analyzátor. Zdrojový soubor pro vizualizaci nebyl poskytnut. Tato práce tedy převzala první čtyři moduly, ze kterých jsou *maintainer* a *crawler* implementovány jak pro původní IPv4, tak i dodatečně pro IPv6 protokol. Moduly *maintainer, crawler* a *injektor* jsou rozebrány v této podkapitole, analyzátor je popsán v podkapitole 6.6.

6.3.1 Maintainer

Maintainer je program, který se spouští před samotným agentem. Jeho cílem je vytvořit seznam aktivních uzlů, ze kterých vybírá určitou podmnožinu uzlů a tu poté ukládá do souboru, kde každý soubor obsahuje 1000 aktivních uzlů. Tyto soubory slouží jako prostředek pro komunikaci mezi komponentou *maintainer* a *crawler*. K ukládání do souboru dochází v pravidelných intervalech, který byl oproti původní práci zkrácen na 10 sekund. Důvodem zkrácení časového intervalu pro vytvoření souboru s uzly byla snaha o snížení časových nároků na experiment. Také zde pravidelně dochází k obměně seznamu aktivních uzlů, aby nedocházelo k poskytování duplicitních dat. Tento modul tvoří třída *Maintainer*, která dědí z abstraktní třídy *AbstractCrawler* a implementuje shromažďování a poskytování IPv4 i IPv6 uzlů. Důsledkem nashromáždění uzlů jsou lepší výsledky dosažené monitorovacím agentem.



Obrázek 6.2: Architektura monitorovacího systému z práce od Wanga a Kangasharju [14].

6.3.2 Agent

Monitorovací agent je v původní práci označen jako *crawler*. V této práci se však nachází *DHT crawler*, a proto byl pro lepší odlišení zvolen název *agent*. Vstupem tohoto programu mohou být soubory vytvořené programem *maintainer* sloužící pro lepší střádání uzlů (bootstrapping). Jádrem implementace je třída *NodeCrawler*, která opět dědí z abstraktní třídy *AbstractCrawler*.

Činnost tohoto modulu spočívá ve střádání uzlů, které je již popsané v podkapitole 6.2.1. Hlavní rozdíl oproti DHT crawleru spočívá ve dvou implementačních odlišnostech:

1. V metodě *start_sender()*. Úkolem této metody je stejně jako u všech ostatních tříd zasílání zpráv *find_node*. Zde se však tato metoda liší v tom, že v případě klasifikace uzlu jako uzlu ze stejné n-bitové zóny je ID uzlu pozměněno. Změna se provádí v cyklu, kde je v každé iteraci provedena změna ID a následně je zaslána zpráva *find_node* i na toto změněné ID. Změna ID se realizuje na pozicích několika nejméně hodnotných bitů (*least significant bits*), čímž je docíleno toho, že nové ID opět spadá do stejné n-bitové zóny. Tímto je zvětšena šance na nalezení více uzlů v požadované zóně.

2. V metodě *convergeSpeed()*. Tato metoda u každého nového uzlu zjišťuje, zda jeho ID spadá do stejné n-bitové zóny jako je ID crawleru. V pozitivních případech je uzel klasifikován jako uzel patřící do stejné zóny a je inkrementován čítač pro počet uzlů v dané zóně.

Výsledkem tohoto modulu je opět soubor obsahující informace o uzlech, který má název „ipv[4/6]nodes.datum_a_cas.id_uzlu.json“ stejně jako výstup DHT crawleru. Pravidelně jsou také vypisovány informace o aktuálním stavu uzlů, počtu uzlů v dané zóně atd.

6.3.3 Injektované uzly

Zdrojový soubor implementující injektované uzly byl opět převzat z práce od Wanga a Kangasharju [14]. Jádro tvoří třída *Injector*, která jako jediná nedědí z abstraktní třídy *Abstract-*

Crawler. Důvodem je odlišná implementace metod, které jsou implementovány v abstraktní třídě. Tento modul bylo opět zapotřebí rozšířit o implementaci pro IPv6 protokol.

Z funkčního hlediska se jedná o kontrolované uzly, které spadají do konkrétní n-bitové zóny. Tyto uzly jsou po vytvoření vloženy do sítě BitTorrent. Jejich implementace tvoří odlehčenou verzi BitTorrent klienta, který si uchovává svou routovací tabulku a má za úkol pouze ohlásit svou existenci a také odpovídat na dotazující se zprávy *ping* a *find_node*.

Vstupem je pouze textový soubor obsahující seznam 160-bitových ID injektovaných uzlů uložených v datovém typu *long*. Výstup tvoří pouze informativní výpis o aktuální velikosti seznamu uzlů, fronty atd.

6.4 Pomocné zdrojové soubory

Pomocné zdrojové soubory využívané implementovaným systémem lze rozdělit do několika skupin. Do první skupiny spadají čtyři oficiální moduly pro jazyk Python. Konkrétně bencode, common, btdht a requests, kde moduly btdht a requests je zapotřebí dodatečně stáhnout a nainstalovat.

Další skupinu tvoří dva zdrojové soubory:

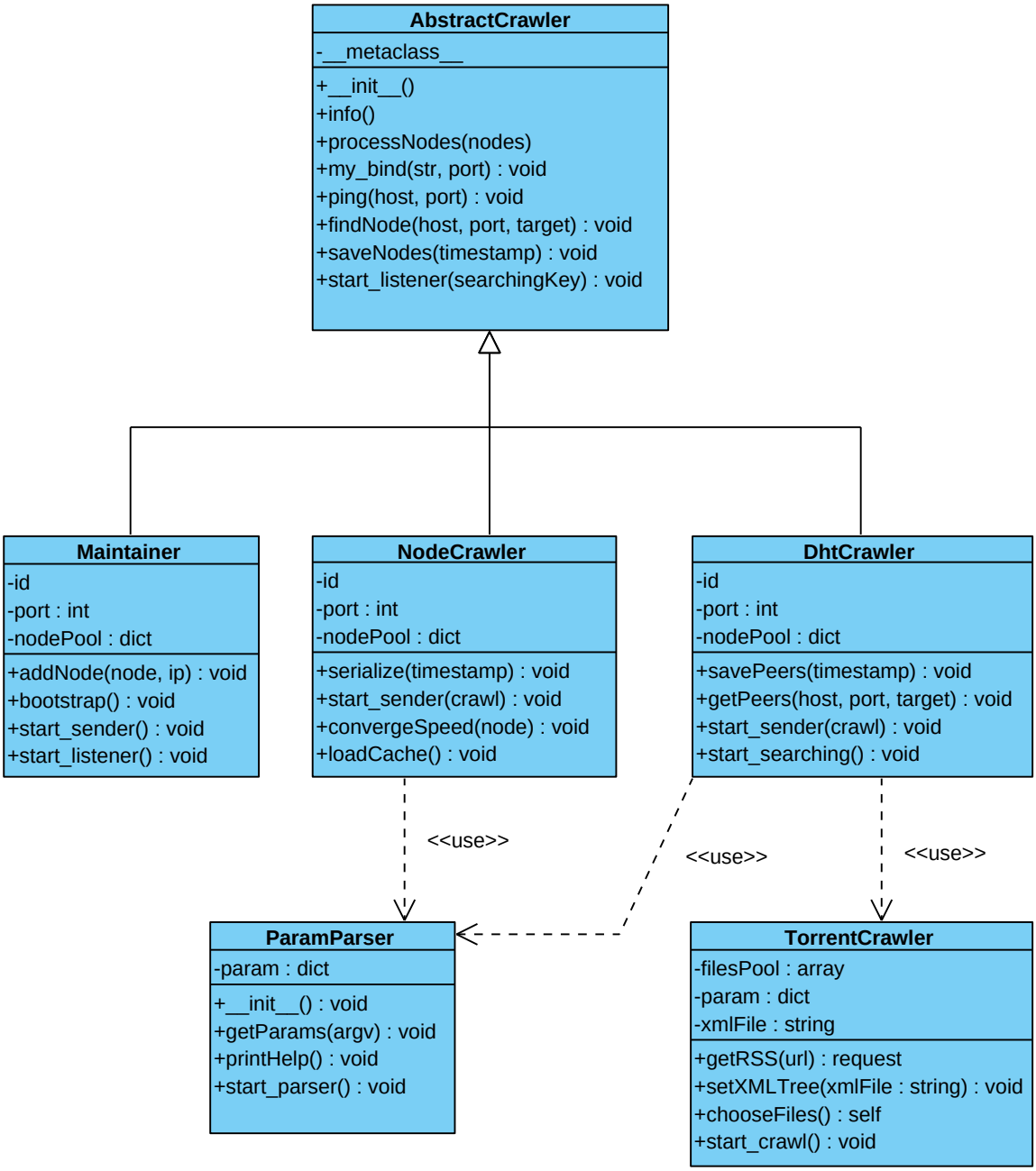
- *khash.py*: Jedná se o zdrojový soubor obsahující metody potřebné pro práci se 160-bitovými hodnotami ID uzlů a infohash souborů jako je metoda pro vytvoření nového ID, převod formátu ID atd. Tento soubor je volně dostupný pod licencí: *BitTorrent Open Source License*.
- *db_utils.py*: Tento soubor obsahuje pomocné funkce všeho druhu. Především obsahuje funkce pro dekódování řetězce obsahující tzv. *kompaktní* formát uzlů (metoda *unpackNodes()*) i peerů (*unpackValues()*). Tento formát vychází ze specifikace protokolu DHT a pro uzly obsahuje v řetězci ID, IP adresu a port. Kompaktní formát pro peera obsahuje pouze IP adresu a port.

Poslední skupinou jsou pomocné třídy, které jsou využity při monitorování:

- *ParamParser*: Vedle samotné funkcionality bylo nutné implementovat zpracování parametrů příkazové řádky a jejich kontrolu. O to se stará třída ParamParser, která je využita oběma přístupy pro monitorování. Výstupem této třídy je slovník obsahující zvolené parametry a jejich hodnoty.
- *TorrentCrawler*: Třída TorrentCrawler slouží jako pomocný modul obstarávající informace o torrent souborech. V první částí se provádí stažení RSS souboru ve formátu xml. Je možné explicitně zadat URL adresu pro stažení RSS souboru, také je možné zadat cestu k již existujícímu souboru nebo není třeba zadávat nic. Pokud není zadán žádný parametr, pak probíhá kontrola existence souboru rss_feed.xml, a poté je případně automaticky stažen soubor pro top 100 filmů ze serveru The Pirate Bay.

6.5 Server

Server je další část monitorovacího systému. Jeho cílem je umožnit integraci více modulů provádějící monitorování systému BitTorrent. Činnost serveru spočívá v přijímání a ukládání zaslaných dat, nebo v poskytování uložených dat. Server se skládá ze dvou částí. První část je databáze samotná a druhou část tvoří rozhraní umožňující přístup k této databázi.



Obrázek 6.3: UML diagram tříd, které jsou ve vztahu s abstraktní třídou AbstractCrawler. Seznam uvedených atributů a metod je pouze jejich vybraná podmnožina. Metody bez návratového typu značí abstraktní metody. Od této abstraktní třídy AbstractCrawler dědí třídy DhtCrawler, NodeCrawler i Maintainer. Mimo to je zde také zobrazen vztah zbylých pomocných tříd ParamParaser i TorrentCrawler.

6.5.1 Datový model

Vzhledem k využití databáze SQLite, je databáze uložena v souboru. Kontrola existence souboru případně jeho inicializace je provedena serverovou částí aplikace při jejím spuštění. Zvolený název databáze je „dht_crawling.db“.

Základní datový model tvoří tři entity:

- Node - Jedná se o uzel v rámci sítě BitTorrent, který může být objeven jak při prohledávání n-bitových zón, tak při vyhledávání peerů pro konkrétní torrent soubory. Tato entita má sloupce nodeID (160-bitový identifikátor uzlu), type (typ - 4/6), host (IP adresa), port, timestamp (časové razítko).
- Peer - Jedná se o konkrétního BitTorrent klienta, který se účastní distribuce torrent souborů. Uchovává sloupečky: host, type, port a timestamp.
- File - Entita uchovávající informace o konkrétních sdílených souborech. Tato tabulka obsahuje pouze dva sloupečky: infohash (160 bitový identifikátor souboru) a name (název souboru).

Jelikož kardinalita mezi těmito entitami vykazuje vztahy M:N (anglicky „many-to-many“), bylo zapotřebí vytvořit i vazební tabulky pro každou dvojici entit. Vznikly tedy tabulky:

- File_Node - Tato tabulka uchovává vztah mezi uzly a mezi sdílenými torrent soubory. V praxi to znamená, že záznam v tabulce značí jaký uzel uchovává informace o peerech sdílejících tento soubor. Platí, že informace o jednom souboru může poskytovat 0 až N uzlů a zároveň platí, že uzel může uchovávat informace o 0 až N souborech.
- File_Peer - Vztah mezi tabulkou File a Peer určuje, který peer se podílí na sdílení konkrétního souboru. Na sdílení souboru se podílí více peerů a zároveň peer se může podílet na sdílení 1 až N souborů.
- Peer_Node - Tabulka pro vztah mezi peery a uzly určuje, který uzel uchovává, respektive který oznámil daného peera. Opět platí vztah M:N. Uzel tedy oznamuje 0 až N peerů a peer je oznámen 1 až N uzly.

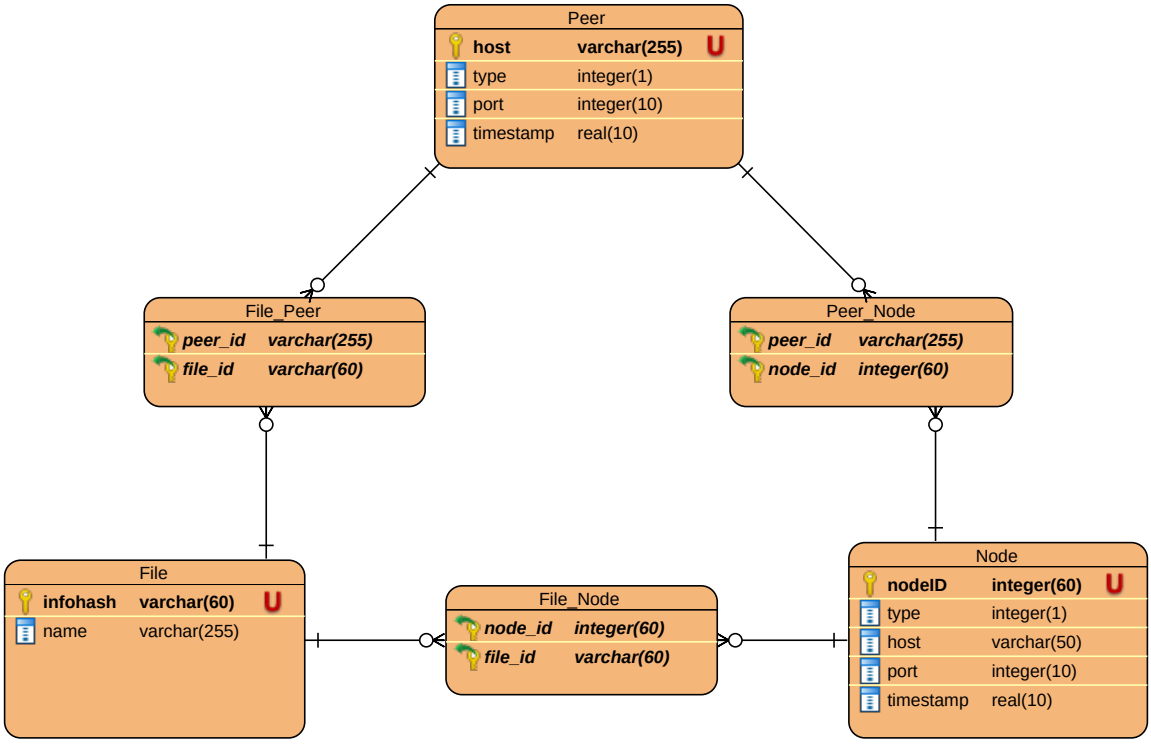
Zde popsané vztahy mezi jednotlivými entitami a vazebnými tabulkami jsou zobrazeny na obrázku 6.4.

6.5.2 Rozhraní pro přístup k databázi

Rozhraní pro přístup k databázi je tvořeno jednoduchou aplikací klient-server. Serverová část aplikace běží na Ubuntu serveru zpřístupněném přes protokol IPv6. Proto je pro komunikaci vytvořen program server, který zasílá data přes IPv6 sockety. Aplikace klient-server je implementována ve verzi jazyka Python 3, jelikož zde již není závislost na přejatých zdrojových souborech ve starší verzi jazyka.

Třídy

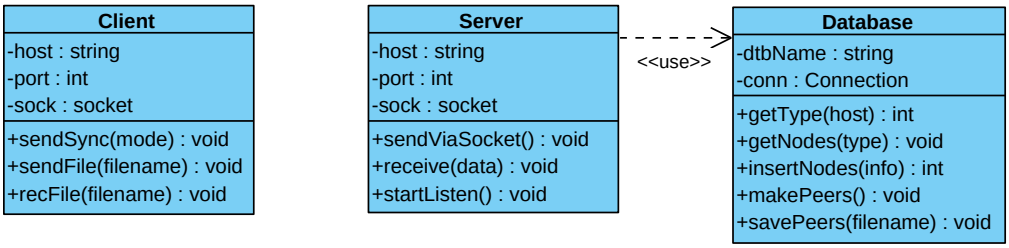
Jádro rozhraní, respektive serveru tvoří třída *Server*, která obsahuje metody potřebné pro vyřizování požadavků od klienta. Příkladem těchto metod jsou metody: *sendViaSocket* nebo



Obrázek 6.4: ERD diagram zobrazující schéma databáze.

receive. Na straně klienta se vyskytuje třída *Client*, která má velmi podobné metody jako třída server. Rozdíl je v tom, že u něj není potřeba implementovat metodu, která rozhoduje zda se budou data odesílat nebo přijímat. To je zde rozpoznáno podle vstupních parametrů. Naopak nutná je metoda pro zaslání inicializačního socketu zajišťující navázání spojení se serverem.

Třída *Server* má přístup k metodám pro čtení a zápis do databáze. Přístup probíhá skrz třídu *Database*, která implementuje metody pro přístup k databázi.

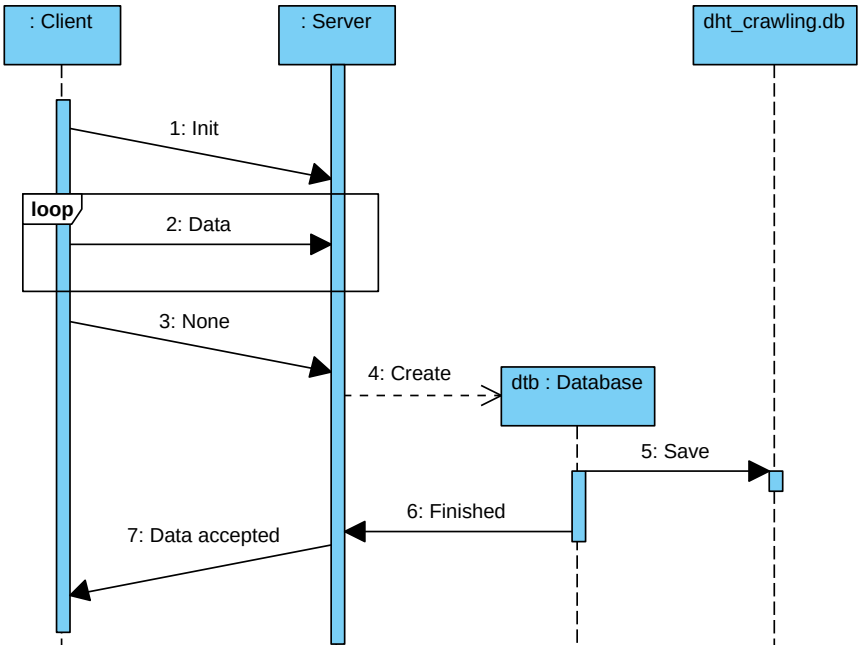


Obrázek 6.5: UML diagram tříd zobrazující třídy v aplikačním rozhraní. V diagramu je pouze určitá podmnožina metod těchto tříd.

Funkcionalita

Funkcionalita představuje využití architektury klient-server, kde klient iniciuje komunikaci a server zpracovává požadavky. Komunikace začíná úvodní zprávou, která na straně serveru slouží k rozpoznání požadavku a provedení příslušné akce (poskytnutí/příjem dat). Typicky se bude klient spouštět nad složkou obsahující data získaná pomocí monitorovacích programů. Výstup serveru je ukládán do logovacího souboru. Funkčnost z pohledu serveru vypadá následovně:

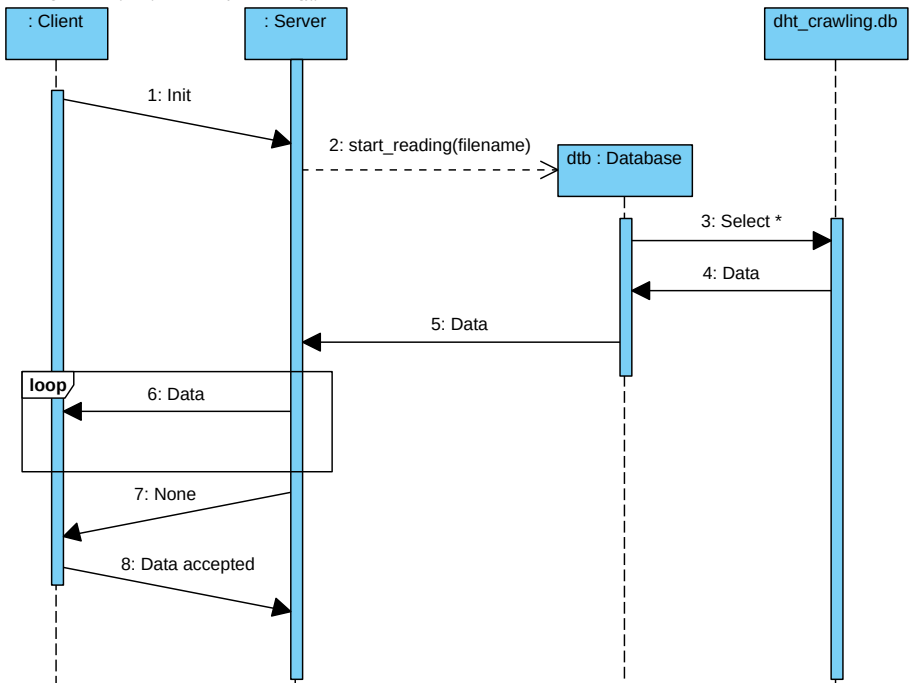
- Příjem dat - První zpráva od klienta začíná řetězcem „-s-soc-“. Písmeno „s“ jako send (z pohledu klienta) a „soc“ značí socket. K tomuto řetězci je zkonkatenován název souboru, do kterého server uloží data. Zpracovány jsou pouze soubory obsahující klíčová slova: „node“, „peer“ a „torrent“. Zpracováním je myšleno načtení JSON souboru (viz podsekce 6.5.2) a uložení získaných dat do databáze (viz obrázek 6.6).



Obrázek 6.6: UML sekvenční diagram s ukázkou komunikace přijímání dat u serveru. Zpráva *Init* může vypadat: „-s-soc-ipv6peers.2018.json“. Následuje zasílání dat od klienta ukončené zprávou *None*. Server poté vytváří instanci třídy *Database*, skrz kterou ukládá data do databáze. Po uložení server oznamuje úspěšné/neúspěšné ukončení zprávou *Response*.

- Poskytování dat - První zpráva zaslána od klienta začíná řetězcem „-r-soc-“. Po řetězci následuje název souboru, který klient žádá. Server zkontroluje, zda je daný název souboru v předem stanoveném seznamu. Pokud ne, odesílá chybové hlášení a ukončuje spojení. V opačném případě volá příslušnou metodu pro čtení potřebných dat z databáze a následně tato data odesílá.

Zprávy *Init* obsahují zkratku *sock*, jelikož byla implementována funkčnost pomocí aplikace NetCat i pomocí socketů. Na straně serveru bylo potřeba rozlišit, kterou variantu klient preferuje, a proto je tento řetězec součástí zprávy. NetCat se však ukázal jako nevhodný pro posílání více souborů a zůstal proto nevyužit.



Obrázek 6.7: UML sekvenční diagram s ukázkou komunikace zasílání dat od serveru klientovi.

Formát JSON souboru

V této práci je využit JSON formát souboru pro manipulaci se získanými daty. Další možností bylo využití tzv. serializace v Pythonu pomocí modulu „pickle“, který je schopný jednoduše ukládat datové objekty. Pro zvolení JSON formátu však vedlo několik skutečností. V první řadě je výsledný soubor menší, lidsky čitelný, bezpečnější a především je přenositelný i na jiné jazyky mimo Python.

V této práci jsou použity tři typy JSON souborů, kde každý je vytvořen z jiného formátu slovníku:

- Pro uzly: Název těchto souborů začíná prefixem „ipv[4/6]nodes“. Tyto soubory jsou výsledkem jak zónových agentů, tak i DHT crawleru, který před samotným hledáním torrent souborů sbírá určité množství uzlů pro rychlejší hledání. Výsledný tvar slovníku vypadá:

  { ’nodeID1’ : { ’timestamp’: hodnota, ’host’ : "IP_adresa1", ’port’ : port1},
  ’nodeID2’ : { ’timestamp’: hodnota, ’host’ : "IP_adresa2", ’port’ : port2},...},

  kde „nodeIDx“ je 160-bitový identifikátor uzlu.

- Pro peery: Název začíná prefixem „ipv[4/6]peers“. Vznikají pouze u DHT crawleru a obsahují získané informace o peerech podílejících se na sdílení určitých souborů. To znamená, že zde jsou uvedeny vazby mezi jednotlivými soubory, uzly a peery. Formát slovníku vypadá následovně:

  {infohash1: {"peers": [], "nodes": [], "name": nazev1},
  infohash2: {"peers": [], "nodes": [], "name": nazev2}, ... },

kde „infohash“ je 160-bitový identifikátor konkrétního souboru, „peers“ značí seznam peerů, „nodes“ seznam uzlů a „name“ je název souboru získaný typicky z RSS souboru. Seznam „peers“ je pole peerů, respektive slovníků mající formát:

{"timestamp": hodnota, "addr": ["IP_adresa", port]}

Prvek v seznamu uzlů má následující položky:

{"timestamp": hodnota, "nodeID": "nodeIDx", "nodeAddr": ["IP_adresa", port]}

- Pro torrenty: Tento soubor vzniká opět pouze u DHT crawleru a jedná se o seznam dvojic (name,infohash) obsahující základní informace o torrent souborech.

6.6 Zpracování dat

Podkapitola zpracování dat se věnuje načítání a především pak analýzou dat získaných pomocí monitorovacích modulů. V první řadě je zde popsán výše zmíněný převzatý zdrojový soubor *analyzer* (viz podkapitola 6.3). V další části je popsán skript pro vyhodnocení prohledávání BitTorrent sítě s injektovanými uzly.

6.6.1 Analyzer

Poskytnutý zdrojový soubor *analyzer* je jednoduchý skript, jehož úkolem je procházení seznamu nashromážděných uzlů případně IP adres za účelem provedení lokalizace dané adresy. Jádro tvoří třída Parser, která implementuje metody pro získání potřebných informací o rozpoložení lokací jako jsou státy a města. Tyto metody jsou využívají metody třídy IPDB, které slouží pro převod IP adresy do formátu *integer* a pro následné poslání dotazu do databáze „ip.db“. Tato databáze pochází od společnosti Hexasoft Development Sdn. Bhd. a spadá pod produkt s názvem IP2Location LITE Edition[1]. Tento produkt obsahuje volně dostupné varianty databází sloužící pro zpracování IP adres. Databáze samotná obsahuje intervaly IP adres převedených do formátu *integer*, což umožňuje ověření, zda IP adresa patří do daného intervalu adres. Dále pak je pro tento modul podstatná informace o státu, jeho zkratce a o městě, které přísluší k danému intervalu adres.

Původní skript obsahoval pouze zpracování informací o uzlech, které byly navíc poskytnuty ve formátu „pickle“. Proto došlo ke přizpůsobení změnou dat na formát JSON. Dále pak bylo zapotřebí, aby skript rozlišoval, zda se jedná o uzly nebo o peery. Důvodem je odlišný formát uložení peerů a uzlů. Dalším nedostatkem původního skriptu bylo zaměření pouze na IPv4 adresy. Proto došlo k rozšíření o implementaci analýzy i pro IPv6 adresy. Není však zapotřebí explicitně zadávat, zda se jedná o IPv4 nebo IPv6 adresu. Skript si tuto informaci zjistí sám.

Výsledkem analyzátoru je výpis rozpoložení účastníků BitTorrentu do světových lokací, respektive států a měst.

6.6.2 Evaluator

Skript nazvaný jako *evaluator* slouží pro sumarizaci dat nasbíraných pomocí DHT crawleru. Implementace je skryta v třídních metodách třídy *Evaluator*, která obsahuje metody čtení souborů, pro získání dat ze souborů a následný výpis nashromážděných dat.

[1] Dostupné na adrese: http://lite.ip2location.com

Mezi data, která tento skript sumarizuje, patří data týkající se objevených uzlů, získaných peerů a redukcí prováděných nad získanými peery:

1. Data k objeveným uzlům: celkový počet objevených uzlů, procentuální vyjádření podílu přijatých odpovědí ku odeslaným dotazům nebo např. procentuální vyjádření duplikací v ohlašovaných uzlech.

2. Data týkající se peerů: Ohledně peerů se sledují informace typu: počet sledovaných torrent souborů, celkový počet nahlášených peerů, počet duplikací v rámci konkrétních souborů včetně procentuálního vyjádření atd.

3. Data týkající se filtrování: V rámci filtrování se pozoruje: počet uzlů redukovaných z důvodu ohlášeného portu s hodnotou 1 včetně výpočtu v procentech a počet uzlů odstraněných po ověřování aktivity uzlu pomocí zprávy ping taktéž zahrnující procentuální vyjádření.

Tento skript bude využit při manuálním vyhodnocení výsledků, kde bude aplikován na složky obsahující data získaná v určité dny nebo například v určitých částech dne pro porovnání aktivity této sítě v různých denních intervalech. Další možností využití je aplikace na složky obsahující pouze data s IPv4 případně IPv6 peery pro umožnění jejich snadného porovnání.

Evaluator je doplněn skriptem *Summarizer*, který prohledává všechny výstupy DHT crawlerů v dostupných složkách a následně vypíše získané hodnoty pro všechna měření i pro jednotlivé moduly (btdht, IPv4 Dht Crawler i IPv6 Dht Crawler).

6.6.3 Pomocné skripty pro zónové agenty

Pomocné skripty pro zónové agenty mají za úkol usnadnit provádění experimentu podle práce od Wanga a Kangasharju [14]. Patří sem skripty:

- *Experiment setter* - Je skript, který usnadňuje spuštění experimentu pro zjištění přibližné velikosti sítě. Vznikl za účelem přípravy hodnot ID pro samotný experiment. V první řadě se generuje ID pro samotné crawlery (zónové agenty), kde je celkem vygenerováno pět ID ve 12-bitové zóně. Následuje vygenerování x hodnot ID, kde jedno ID je ID jednoho konkrétního injektovaného uzlu. V tomto případě je x zvoleno dle experimentu [14] na hodnotu 50. Tyto ID jsou poté vepsány do souboru, jehož název tvoří i ID uzlu. Všechny hodnoty ID jsou ve formátu long.

- *Finder* - Jedná se o skript provádějící zpracování výstupních souborů. Skript prochází složku obsahující výstupy spuštěných crawlerů obsahující nasbírané uzly a bývá spouštěn po ukončení jednoho měření. Zpracovává výstup od pěti crawlerů (viz podkapitola 7.2.2). Jeho úkolem je sjednocení množin uzlů, které vznikly spouštěním pěti crawlerů, a následné prohledávání sjednocené množiny za účelem vyhledání ID injektovaných uzlů. Výstupem skriptu je výpis obsahující seznam nalezených injektovaných uzlů včetně jejich celkového počtu pro dané měření.

- *P_count* - *P_count* je skript procházející výstupy skriptu *Finder* za účelem výpočtu hodnoty p, která značí pravděpodobnost nalezení uzlu v síti. Skript prochází výstupy všech měření ohledně experimentu (IPv4 i IPv6) a počítá hodnotu p včetně směrodatné odchylky.

Kapitola 7

Testování a vyhodnocení

Kapitola testování a vyhodnocení se věnuje ověřování funkcionality a vyhodnocení výsledků dosažených softwarem popsaným v kapitole 6. První část se zaměřuje na *DHT crawler* a jsou zde popsány výsledky týkající se počtu získaných peerů, množství torrentů bez nalezených peerů, délce běhu a v neposlední řadě také porovnání tří monitorovacích modulů (pro btdht, IPv4 i pro IPv6 crawler). Druhá část této kapitoly se věnuje vyhodnocení experimentů prováděných na základě článku od Wanga a Kangasharju [14], jejichž cílem je zjistit přibližnou velikost sítě BitTorrent. Také v této podkapitole proběhlo měření jak pro IPv4, tak i pro IPv6 protokol. Veškeré testování probíhalo v reálné síti. Bod zadání týkající se testování na vlastním trackeru je splněn implementací crawlerů a agentů jakožto uzlů v rámci BitTorrent sítě. Vychází se ze skutečnosti, že uzly využívající protokol Mainline DHT plní vedle role uzlu i roli trackeru.

7.1 DHT crawler

Tato podkapitola popisuje jednotlivá měření. V každém měření jsou získávána data od všech tří modulů (btdht, IPv4 a IPv6). Mezi sledované výstupy patří:

- Počet sledovaných torrent souborů.
- Počet torrent souborů, pro které nebyl nalezen žádný peer.
- Celkový počet ohlášených peerů.
- Počet unikátních peerů v rámci jednoho interesovaného souboru.
- Počet odfiltrovaných peerů.
- Doba běhu programu.

Tato měření byla prováděna vždy pro deset vyhledávaných torrent souborů a každé probíhalo v pěti různých bězích pro přesnější statistické vyhodnocení. Vyhledávané torrenty byly získávány ze dvou RSS souborů, kde první obsahuje soubory z kategorie top 100 filmů a druhý obsahuje top 100 unixových souborů ze serveru Pirate Bay. Příkladem sledovaných unixových souborů patří: Tails v3.6 the amnesic incognito live system, Red Hat Enterprise Linux (RHEL) Server 7.0 x86-64 Multilingual, Kali Linux Amd64, [Iso - MultiLang] a další. Rozdíl mezi soubory z obou RSS souborů je především v jejich stáří (viz tabulka 7.1). Měření vypadají následovně:

1. Nejlepších 10 Filmů - Hledání peerů pro prvních 10 nejoblíbenějších filmů (zkráceně Top 10 F.).

2. Posledních 10 Filmů - Hledání peerů pro posledních 10 nejoblíbenějších filmů (zkráceně Last 10 F.).

3. Top 10 Unixových souborů - Hledání peerů pro prvních 10 nejoblíbenějších unixových souborů. Stáří těchto souborů je znatelně vyšší než v předchozích dvou měřeních. Toto měření má za úkol zjistit, zda má stáří torrent souboru vliv na množství peerů podílejících se na jeho sdílení (zkráceně Top 10 U.).

Hypotézy pro uvedená měření:

1. Měření *Top 10 F.* bude vykazovat největší úspěšnost nalezení peerů, největší množství nalezených peerů a také i nejmenší časovou náročnost.

2. Měření *Top 10 U.* bude vykazovat nejhorší výsledky ohledně procentuální úspěšnosti nalezení peerů, množství peerů i časové náročnosti.

3. V BitTorrent síti využívá protokol IPv4 větší množství peerů než protokol IPv6.

	Top 10 F.	Last 10 F.	Top 10 U.
Nejnovější torrent	10	10	45
Nejstarší torrent	124	1386	3726
Průměrné stáří	61	319	926

Tabulka 7.1: Stáří souborů v rámci jednotlivých měření. Stáří je uvedeno v počtu dní od data publikace (ke 28. 4. 2018).

7.1.1 Celkové porovnání

Celkové porovnání popisuje výsledky dosažené opakovaným měřením jednotlivých modulů (btdht, IPv4 a IPv6) pro jejich snadné porovnání.

1. **btdht crawler**

   Z pohledu úspěšnosti nalezení peerů pro konkrétní torrent je nejspolehlivější modul btdht s průměrnou úspěšností 80,67% (100 - 19,33). Tento modul také na rozdíl od modulů implementovaných v této práci funguje na principu „hledej dokud nenajdeš“, čímž je sice zvýšena úspěšnost nalezení peerů, ale také tímto roste časová náročnost vyhledávání. Úspěšností i počtem nalezených peerů se zdá, že modul btdht vyvrací hypotézy č. 1 a 2. Avšak nárůst časové náročnosti, který lze pozorovat v posledním řádku tabulky 7.2 tyto hypotézy naopak potvrzuje. Mimo to lze pozorovat odchylku v úspěšnosti vyhledávání pro 2. měření (Last 10 F.). Tato odchylka může být způsobena např. tím, že byla pro tato prohledávání vygenerována ID crawleru s menší vzdáleností k infohashům daných souborů než ve zbylých měřeních. Dalším teoreticky možným důvodem této odchylky může být změna oblíbenosti, respektive počtu peerů v čase. Tato měření byla prováděna nad konkrétním RSS souborem, který nebyl v průběhu měření aktualizován a tudíž mohla popularita souborů různě kolísat.

	Top 10 F.	Last 10 F.	Top 10 U.	Průměr
Počet sledovaných torrent souborů	400	400	400	400
Počet souborů bez nalezených peerů	80	69	83	77,33
Podíl souborů bez nalezených peerů [%]	20,00	17,25	20,75	19,33
Celkový počet ohlášených peerů	4048	4430	3873	4117
Průměrný počet peerů na 1 soubor (bez souborů s 0 peery)	12	13	12	12
Průměrná doba běhu na 1 soubor [sec]	4,22	5,10	6,86	5,39

Tabulka 7.2: Výsledky modulu btdht crawleru.

2. **IPv4 DHT crawler**

   Modul DHT crawleru pro IPv4 vykazuje nejnižší procentuální úspěšnost (50,25%) nalezení peerů. Hlavním důvodem je skutečnost, že IPv4 i IPv6 moduly používají stejný přístup k prohledávání sítě. Výsledkem je razantní omezení první fáze prohledávání sítě za účelem nastřádání uzlů (viz podkapitola 6.2.2), čímž byla efektivita IPv4 crawleru snížena. Tato redukce však výrazně zvedla úspěšnost IPv6 modulu, a proto byl ponechán tento kompromis. Ukazuje to však na mírné odlišnosti nad protokolem IPv4 a IPv6. Bylo by tedy vhodnější mít pro tyto protokoly samostatné vyhledávací algoritmy.

   Naměřené výsledky modulem IPv4 crawleru však potvrzují hypotézy č. 1 i 2. To lze pozorovat v tabulce 7.3 především v řádcích týkajících se úspěšnosti nalezení peerů a množství nalezených peerů, kde lze pozorovat klesající tendenci výsledků od měření *Top 10 F.* k *Top 10 U.* Tento jev se odráží také v počtu získaných uzlů, který reflektuje větší snahu modulu o nalezení peerů a nejvyšší je právě u 3. měření (Top 10 U.).

   Výhodou tohoto modulu je fakt, že vykazuje vysoké počty nalezených peerů pro všechny torrenty. Jestliže se vyřadí torrenty bez nalezených peerů, pak je průměrný počet peerů nalezených IPv4 modulem mnohonásobně vyšší než u zbývajících modulů. V prvním měření (*Top 10 F.*) je průměrný počet ohlášených peerů na soubor dokonce 225, což činí průměrně 77 nalezených peerů za sekundu oproti necelým 3, které poskytuje btdht modul.

	Top 10 F.	Last 10 F.	Top 10 U.	Průměr
Průměr nalezených uzlů na 1 měření	32959	34158	33996	33713
Počet sledovaných torrent souborů	400	400	400	400
Počet souborů bez nalezených peerů	176	192	229	199
Podíl souborů bez nalezených peerů [%]	44,00	48,00	57,25	49,75
Celkový počet ohlášených peerů	50407	33388	10219	31338
Celkový počet peerů po filtrování	45833	31306	9939	29026
Procento peerů po filtrování [%]	90.93	93.85	97,26	94,01
Celkový počet peerů po pingu	5856	3743	1326	3642
Procento peerů po pingu [%]	12,78	11,96	13,34	12,69
Průměrný počet peerů na 1 soubor (bez souborů s 0 peery)	225	160	59	148
Průměrná doba běhu na 1 soubor [sec]	2,90	3,28	3,27	3,15

Tabulka 7.3: Výsledky modulu DHT crawleru pro IPv4.

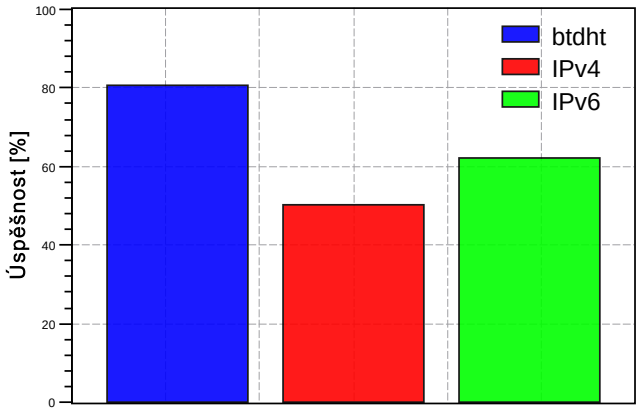
3. **IPv6 DHT crawler**

   Konečně výsledky modulu pro protokol IPv6. Úspěšnost nalezení peerů pro torrent soubor činí 62,17%. Jedná se tedy o větší úspěšnost než IPv4 modul se stejným vyhledávacím algoritmem. Tento modul také potřebuje méně uzlů pro vyhledávání peerů. Z výsledků modulu IPv6 crawleru lze opět pozorovat potvrzení 1. a 2. hypotézy. Zde jsou rozdíly mezi jednotlivými měřeními daleko znatelnější. V první řadě je zde nižší úspěšnost měření týkající se starších unixových souborů, s čímž koresponduje i větší množství nalezených uzlů u tohoto měření podobně jako u IPv4. Dále je také velký rozdíl mezi jednotlivými měřeními z pohledu nalezených peerů. Rychlost měření *Top 10 F.* je průměrně 21 ohlášených peerů na soubor, což činí téměř 9 objevených peerů za sekundu.

	Top 10 F.	Last 10 F.	Top 10 U.	Průměr
Průměr nalezených uzlů na 1 měření	9688	8864	11214	9922
Počet sledovaných torrent souborů	400	400	400	400
Počet souborů bez nalezených peerů	87	142	225	151
Podíl souborů bez nalezených peerů [%]	21,75	35,50	56,25	37,83
Celkový počet ohlášených peerů	6809	3150	606	3521
Celkový počet peerů po filtrování	4870	2642	605	2705
Procento peerů po filtrování [%]	71,52	83,87	99,83	76,84
Celkový počet peerů po pingu	422	263	149	278
Procento peerů po pingu [%]	8,67	9,95	24,63	10,28
Průměrný počet peerů na 1 soubor (bez souborů s 0 peery)	21	12	3	12
Průměrná doba běhu na 1 soubor [sec]	2,39	2,22	2,33	6,94

Tabulka 7.4: Výsledky modulu DHT crawleru pro IPv6.

Výsledky popsané výše potvrzují 1. a 2. hypotézu. To je způsobeno četností uživatelů, kteří sdílejí konkrétní soubory. Je vidět, že stáří souborů tedy ovlivňuje množství peerů podílejících se na jejich sdílení. Dále je také možné pozorovat rozdíly v množství získaných peerů v síti IPv4 a IPv6, což koresponduje s hypotézou č. 3.

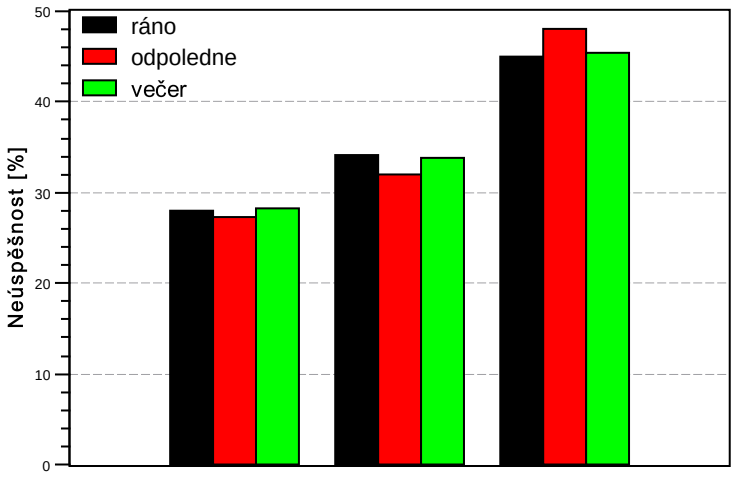


Obrázek 7.1: Úspěšnost jednotlivých modulů pro nalezení peerů pro torrent soubor.

Vedle již popsaných výsledků bylo u modulů DHT crawleru také pozorováno filtrování ohlášených peerů. Jak bylo popsáno v kapitole 6.2.2, probíhají dvě různá filtrování. První odstraňuje peery s hodnotou portu 1 a druhá část má za úkol ověření aktivity daného peera pomocí zprávy ping. Tato metoda filtrování však redukovala 87,31% peerů u IPv4 a u IPv6 dokonce 89,72%, proto je tato metoda filtrování pouze pro informaci a zmíněné adresy nejsou mazány. U modulu btdht nebylo filtrování testováno kvůli větším časovým nárokům na vyhledávání. Při pokusu, který zahrnoval stejné filtrování u btdht modulu, byly získány podobné hodnoty okolo 85% redukovaných peerů. Vysoká míra redukce může být způsoben více jevy. V první řadě se může jednat o klienta, který nemá implementovánu odpověď na zprávu ping. Dále se může jednat o peery, které již nejsou aktivní na síti, a proto neodpovídají. Jedním z dalších důvodů může být tzv. NAT traversal, kde můžou být počítače schovány za dynamickým systémem NAT, na které může dorazit daná zpráva ping, ale směrovač nemusí znát cílovou adresu, a proto na tuto zprávu neodpovídá.

7.1.2 Porovnání v průběhu dne

Porovnání v průběhu dne má za úkol zjistit odlišnosti sítě za pomocí provádění měření v různých fázích dne. Konkrétně byly provedeny měření ráno (mezi 8. a 10. hodinou), odpoledne (mezi 14. a 17. hodinou odpoledne) a večer (mezi 20:00 a půlnocí) středoevropského času (SEČ). Porovnání se provádí podle úspěšnosti nalezení peerů pro torrent a podle průměrného počtu peerů na 1 torrent.

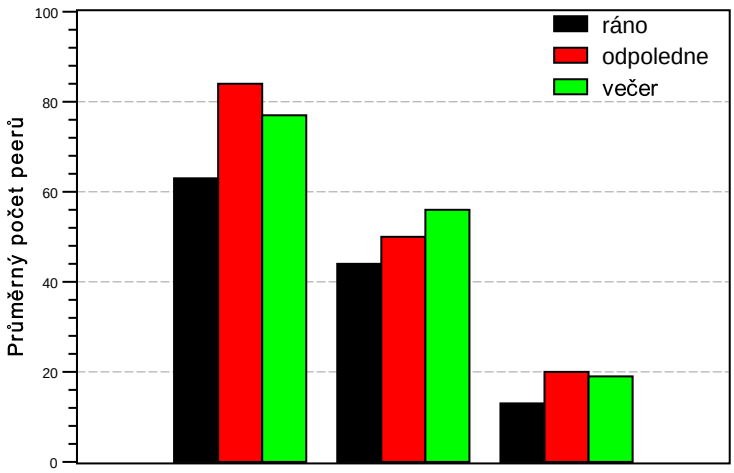


Obrázek 7.2: Procento torrentů bez nalezených peerů. Každá trojice sloupců znázorňuje jedno měření. Zleva: Top 10 F., Last 10 F., Top 10 U.

První otázka zní: „Je neúspěšnost nalezení peerů pro torrent soubor různá v průběhu dne?“. Výsledek testu je zobrazen na obrázku 7.2. Je možné pozorovat, že neúspěšnost nalezení peerů v rámci jednotlivých měření není výrazně závislá na denní době.

Další test se zabývá otázkou, zda dochází ke změně počtu peerů v průběhu dne. Je zde tedy porovnán průměrný počet peerů na jeden torrent v rámci jednotlivých měření (viz obrázek 7.3). První měření, respektive měření u nejoblíbenějších filmů, vyznívá nejlépe pro odpolední měření. Ve večerních hodinách zase přebírá otěže 2. měření. Ohledně celkového porovnání je možné pozorovat relativně podobné hodnoty odpoledne a večer, to už však neplatí pro ranní hodiny. V ranních hodinách bylo získáno nejmenší množství peerů v rámci

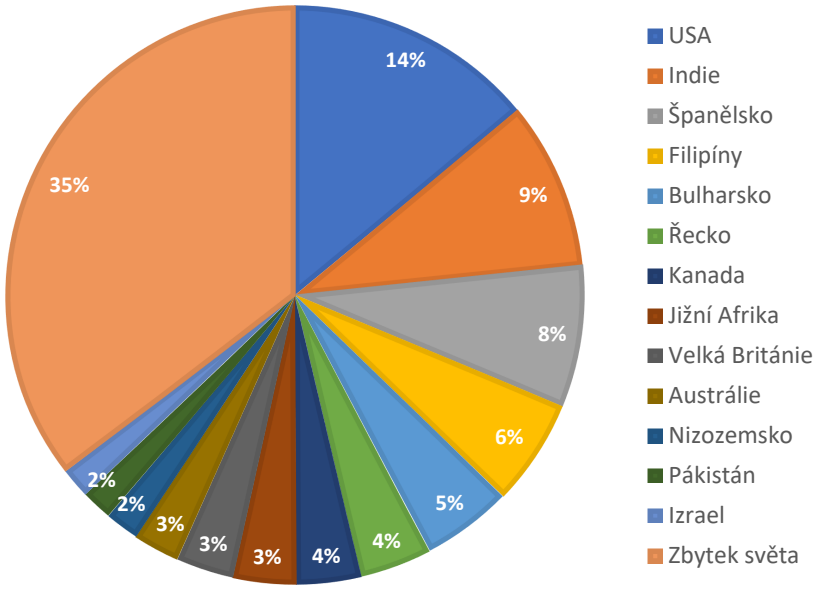
všech měření. Množství peerů, se kterou souvisí i velikost sítě, se tedy v průběhu dne vyvíjí. To může být způsobeno např. různou geografickou polohou jednotlivých skupin uživatelů.



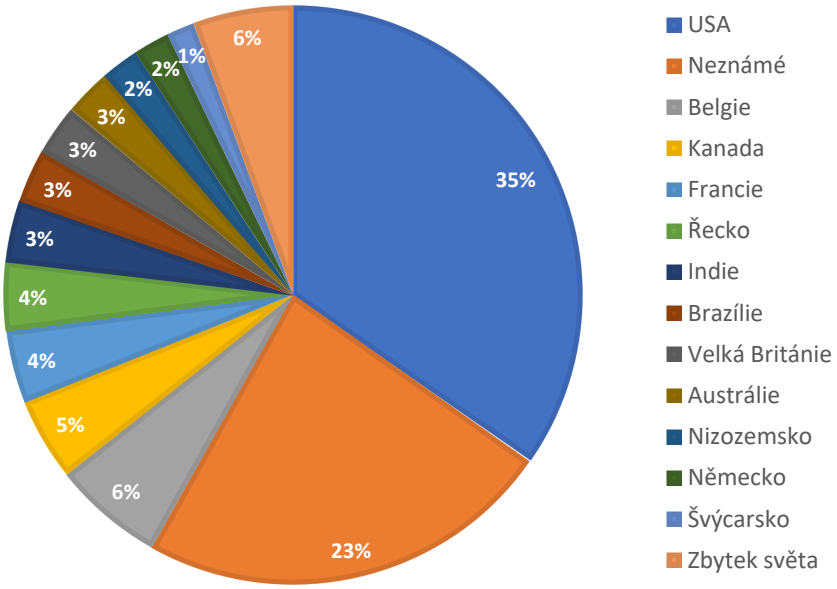
Obrázek 7.3: Graf zobrazující průměrný počet peerů na jeden torrent v průběhu dne. Každá trojice sloupců znázorňuje jedno měření. Zleva: Top 10 F., Last 10 F., Top 10 U.

7.1.3 Geografické rozpoložení

Zpracování dat z pohledu geografického rozpoložení se zaměřilo na data získaná během měření 29. dubna 2018 v odpoledních hodinách. Výsledkem jsou dva grafy zobrazující zastoupení jednotlivých států, kde první zobrazuje IPv4 adresy a druhý IPv6 adresy.



Obrázek 7.4: Graf znázorňující procentuální geografické rozložení při IPv4 měření.



Obrázek 7.5: Graf znázorňující procentuální geografické rozložení při IPv6 měření.

Z grafů 7.4 a 7.5 lze pozorovat, že největší procento zastupují peery, které byly lokalizovány na území USA. U grafu týkající se IPv4 je vidět rovnoměrnější rozpoložení po celém světě, zatímco graf 7.5 vykazuje větší rozdíly a pouze relativně malé procento peerů patří do zbytku světa. U IPv6 lze také na předních pozicích pozorovat vyspělejší státy jako Belgii, Kanadu, Francii oproti IPv4, kde se objevují méně vyspělé státy jako Indie, Filipíny, Bulharsko atd.

7.2 Zónový agent

Vyhodnocení zónových agentů je rozděleno na tři části. První část se věnuje experimentu, ze kterého vycházejí tyto zdrojové soubory. To zahrnuje nejen jeho podrobnější popis, ale také způsob získání a následné využití výsledků. Druhou část tvoří provedení samotného experimentu pro protokol IPv4 a porovnání dosažených výsledků. Třetí část poté tvoří výsledky získané pro protokol IPv6.

7.2.1 Experiment

Cílem experimentu je získat hodnotu p, která udává pravděpodobnost nalezení uzlu. Jelikož existují dvě varianty nalezení uzlu, nalezen a nenalezen, lze z této hodnoty získat pravděpodobnost toho, že daný uzel nebude nalezen jako: $1 - p$. Experiment zavádí pojem „faktor opravy“ CF (anglicky *Correction Factor*), který se vypočítá jako:

$$CF = \frac{1}{p} \tag{7.1}$$

Experiment také určuje interval velikosti sítě, který je určen hodnotou faktoru opravy a odchylkami. Nechť σ je hodnota směrodatné odchylky, M velikost sítě a m ustanovený

odhad velikosti na základě 12-bitové zóny, pak M patří do intervalu $[CF_1 * m, CF_2 * m]$, kde výpočet faktorů vypadá následovně [14]:

$$CF_1 = \frac{1}{p + \sigma} \qquad CF_2 = \frac{1}{p - \sigma} \tag{7.2}$$

7.2.2 IPv4

Provedení experimentu pro protokol IPv4 probíhalo stejně jako u Doe et al. [14]. Do sítě bylo vloženo 50 uzlů patřící do stejné 12-bitové zóny. Následně bylo spuštěno pět crawlerů s ID patřícím také do shodné 12-bitové zóny. Úkolem crawlerů bylo prohledávání sítě BitTorrent za účelem nalezení vložených uzlů. Podle úspěšnosti nalezení je pak vypočítána hodnota p a také směrodatná odchylka. Toto probíhalo v pěti až deseti opakováních a vždy pro jinou 12-bitovou zónu. Každý vložený uzel měl v paměti při spuštění crawlerů okolo 20000 objevených uzlů. Každý crawler běžel přibližně 9 sekund a za tu dobu nastřádal průměrně 29000 uzlů a počet uzlů ve stejné 12-bitové zóně se pohyboval okolo 9000. Výpočet směrodatné odchylky a hodnoty p pro jedno měření je možné vidět v tabulce 7.5.

měření	Počet vložených uzlů	počet nalezených uzlů	roptyl	rozptyl^2
1	50	42	1,1	1,21
2	50	40	3,1	9,61
3	50	39	4,1	16,81
4	50	37	6,1	37,21
5	50	41	2,1	4,41
6	50	45	1,9	3,61
7	50	46	2,9	8,41
8	50	46	2,9	8,41
9	50	46	2,9	8,41
10	50	49	5,9	34,81
průměr		43,1		13,29
směrodatná odchylka				3,65

Tabulka 7.5: Určení směrodatné odchylky.

Vypočtená hodnota p pro toto konkrétní měření tedy činí: $0,862$ se směrodatnou odchylkou $3,65$, to jest: $7,3\%$ (z 50 hledaných uzlů), respektive $0,073$.

V rámci všech provedených měření byla naměřená hodnota $p = 0,8633$ se směrodatnou odchylkou $3,79$. To je $7,58\%$, tedy $0,0758$. Interval s opravujícími faktory, do kterého bude spadat výsledná velikost sítě tedy vypadá následovně:

$$[1,065 * m; 1,269 * m] \tag{7.3}$$

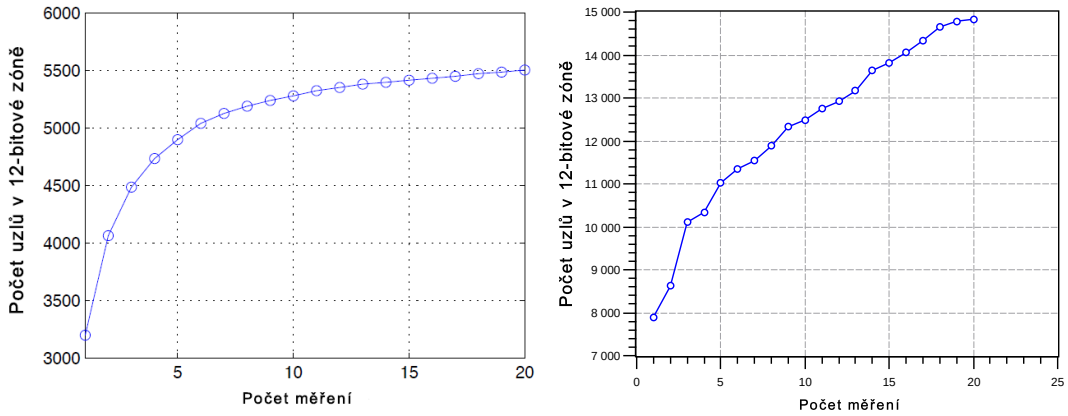
Hodnota m vychází z předpokladu, že uzly jsou rovnoměrně rozloženy mezi jednotlivé 12-bitové zóny. Pak ji lze spočítat jako:

$$m = 2^{12} * 14833 = 60\,755\,968 \tag{7.4}$$

kde hodnota 12 značí měření nad 12-bitovou zónou a hodnota 14833 je počet získaných uzlů z 12-bitové zóně při spuštění 20 agentů (viz obrázek 7.6). Celý interval tedy vypadá:

$$[64\,705\,105; 77\,099\,323] \tag{7.5}$$

Naměřená velikost sítě v době měření se pohybovala okolo 70 milionů uzlů. Na obrázku 7.6 je možné vidět porovnání naměřených dat od 20 různých agentů. Lze zde pozorovat, že měření z této práce vykazují přibližně třikrát vyšší počet uzlů. Jimi naměřená velikost sítě se pohybovala mezi 16-26 miliony uzly. To znamená zhruba třikrát menši velikost sítě oproti zde vypočítané hodnotě.



Obrázek 7.6: Porovnání získaného počtu uzlů ve 12-bitové zóně od více agentů. Vlevo se nacházející graf je převzat od Wanga a Kangasharju [14], graf na pravé straně ukazuje výstup naměřený v této práci v rámci jednoho měření.

7.2.3 IPv6

Experiment pro protokol IPv6 je v principu totožný jako u IPv4. Rozdíl tohoto experimentu je v tom, že injektované uzly se hůře vkládají do sítě, respektive trvá delší dobu, než naleznou a ohlásí se dostatečnému množství uzlů. Proto byl před samotným pokusem pouštěn program *maintainer* pro nalezení úvodních pěti až desíti tisíc uzlů pro lepší začlenění do sítě. To urychlilo proces vložení injektovaných uzlů. Také byl IPv6 crawling doplněn o další zasílání dotazů, bez kterého crawler často skončil po nalezení pár desítek až stovek uzlů. Následně experiment probíhal totožně jako u IPv4. Vzhledem k dosaženým výsledkům tak byla prodloužena doba běhu crawleru, která byla ukončena nalezením 100000 uzlů. IPv6 crawlery běžely průměrně 25 sekund, většina z nich nashromáždila lehce přes 102000 uzlů, kde do stejné 12-bitové zóny patřilo průměrně okolo 17000 uzlů.

Hodnota p ze všech provedených měření má hodnotu $0,6796$. Směrodatná odchylka o hodnotě $4,67$ znamená $9,34\% = 0,0934$. Čili interval hodnoty p je roven $[0,5957, 0,7723]$. Interval s opravujícími faktory, do kterého bude spadat výsledná velikost sítě tedy vypadá následovně:

$$[1,293 * m; 1,706 * m] \tag{7.6}$$

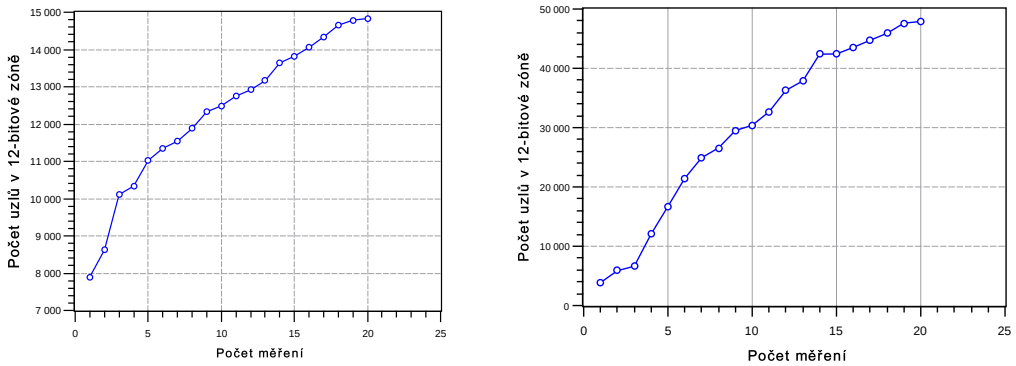
Získaný výsledek ukazuje, že je v IPv6 síti menší pravděpodobnost nalezení uzlu než v síti IPv4. Tato měření také vykazují větší směrodatnou odchylku, čili výsledný odhad nebude tak přesný jako v protokolu IPv4. Fakt, že je hodnota p u IPv6 menší, koresponduje také s obtížnějším bootstrappingem a je způsoben menší velikostí sítě respektive tím, že protokol IPv6 podporuje menší množství klientů. Výše zmíněné úpravy IPv6 crawleru oproti IPv4 zrychlily proces bootstrappingu. Na obrázku 7.7 je zobrazeno porovnání IPv4 a IPv6 crawlerů, na kterém lze pozorovat ono zrychlení zvýšeným počtem dotazování, které se projevilo větším počtem nashromážděných uzlů za stejný čas.

Z obrázku 7.7 je přibližný odhad velikosti sítě v intervalu:

$$[255\,802\,982; 337\,509\,580] \tag{7.7}$$

To by znamenalo, že IPv6 síť by byla přibližně čtyřikrát větší než síť IPv4. To však neodpovídá hypotéze č. 3 popsané v kapitole 7.1, která očekává, že IPv4 síť obsahuje více uzlů než síť IPv6. Tuto hypotézu však výsledky modulů *DHT crawler* potvrdily. Také původní spouštění IPv6 crawlerů ukázalo, že je v IPv6 síti náročné nalézt větší množství uzlů.

Výsledky dosažené experimentem pro IPv6 tedy nejsou uspokojivé. To může být způsobeno odlišným přístupem k prohledávání nebo např. dosud neznámou odlišností IPv6 sítě BitTorrent. Neuspokojivý závěr pro velikost IPv6 sítě dává podnět pro další možné rozšíření.



Obrázek 7.7: Porovnání získaného počtu uzlů ve 12-bitové zóně od IPv4 a IPv6 crawlerů. Vlevo je zobrazen graf pro IPv4 a vpravo pak IPv6. Graf u IPv6 byl vytvořen při stejných podmínkách (limit také 20000 uzlů).

Kapitola 8

Závěr

Cílem této diplomové práce bylo vytvoření monitorovacího systému pro detekci a monitorování peerů sdílejících konkrétní torrenty. Analýza protokolu BitTorrent vyústila v zaměření se na část sítě využívající protokol Mainline DHT (viz kapitola 3). V průběhu analýzy existujících metod byl původní návrh systému monitorující peery rozšířen o monitorovacího agenta, který se zaměřuje pouze na uzly (viz kapitola 4). Návrh systému tedy tvořily dva monitorovací moduly (monitorování peerů, monitorovací agent), implementující dva odlišné přístupy k monitorování sítě BitTorrent. Součástí návrhu se stalo také aplikační rozhraní s databází umožňující spojení implementovaných modulů včetně možnosti rozšíření systému o další nezávislé moduly (viz kapitola 5).

Kapitola 6 se věnovala implementaci daného návrhu, kde oba monitorovací moduly byly rozděleny na více modulů. V rámci monitorování peerů byly implementovány celkem tři samostatné moduly - btdht, IPv4 a IPv6 DHT crawler. Monitorovací agent pro monitorování uzlů se skládá z více zdrojových souborů, které byly převzaty z práce od Wanga a Kangasharju [14] a následně upraveny pro potřeby této práce. Úpravou je myšleno přizpůsobení zdrojových souborů pro umožnění využití abstraktní třídy *AbstractCrawler* (viz obrázek 6.3) a především rozšíření o funkcionalitu pro protokol IPv6. Součástí implementace je také zmíněné aplikační rozhraní s databází.

Závěrem této práce je popis testování a zhodnocení dosažených výsledků (viz kapitola 7). V první řadě zde byla vyhodnocena úspěšnost jednotlivých modulů zabývající se monitorováním peerů a to z pohledu úspěšnosti nalezení peerů pro konkrétní torrent soubor, množství nalezených peerů nebo např. z pohledu rychlosti, respektive počtu získaných peerů za sekundu (viz podkapitola 7.1). Nejspolehlivějším modulem pro vyhledávání peerů s úspěšností nalezení 80,67% je modul *btdht*, který vychází z veřejně dostupné knihovny btdht. Z pohledu množství peerů i rychlosti byl naopak nejlepší modul IPv4 DHT crawler. Výhodou posledního modulu IPv6 DHT crawleru je jeho zaměření na část sítě využívající protokol IPv6, která není tak dobře zmapovaná jako IPv4 část. Ve stejné podkapitole je také porovnáno měření z pohledu denní doby i geografického rozpoložení.

Následující podkapitola 7.2.1 se věnuje vyhodnocení výsledků prováděných experimentů podle článku od Wanga a Kangasharju [14]. Zvlášť je zde popsáno vyhodnocení IPv4 a IPv6. Výsledky pro IPv4 síť odhadují její velikost na zhruba 70 milionů uzlů. Měření pro protokol IPv6 ukázaly tuto síť jako čtyřikrát větší, což neodpovídá měření a počtu nalezených peerů pomocí modulů DHT crawleru. Proto je tento výsledek označen jako neuspokojivý.

Aplikační rozhraní bylo otestováno průběžným zasíláním získaných výsledků jednotlivých modulů. Součástí bylo i zakomponování bakalářské práce od pana M. Vaška [13].

Ta tvoří další nezávislý modul pro monitorování. Obdobně by bylo možné zakomponovat libovolné množství podobných modulů.

Tato práce by mohla být rozšířena více způsoby. Prvním rozšířením, které bylo zmíněné již v průběhu práce, by mohlo být obohacení knihovny *btdht* o podporu Mainline DHT nad protokolem IPv6. Dále by mohly být podrobněji prozkoumány rozdíly v hledání peerů i crawlingu v rámci IPv4 a IPv6 sítě. V této práci byl použit stejný algoritmus pro vyhledávání IPv4 i IPv6 peerů, což se negativně projevilo na úspěšnosti IPv4 DHT crawleru. Naopak u crawlingu bylo při IPv6 prohledávání zapotřebí dodatečné dotazování, a to do značné míry ovlivnilo získané výsledky. Proto by bylo možné se v další práci detailněji zaměřit na specifika IPv6 sítě BitTorrent. V neposlední řadě může být přidán libovolný modul pro monitorování nebo by mohlo být rozšířeno i aplikační rozhraní o různé dotazy nad uloženými daty. Zajímavé rozšíření by mohlo být i vytvoření modulu, který by sledoval vytváření nových torrent souborů a pro každý by hledal tzv. inicializačního peera, respektive uživatele, který tento soubor vložil do BitTorrent sítě.

Literatura

[1] Bauer, K.; McCoy, D.; Grunwald, D.; aj.: *BitStalker: Accurately and Efficiently Monitoring BitTorrent Traffic*. 2009, [Online; navštíveno 18.12.2017].
URL https://ieeexplore.ieee.org/abstract/document/5386457/

[2] Camarillo, G.: *RFC 5694 - Peer-to-peer (P2P) Architectures*. Listopad 2009, [Online; navštíveno 12.10.2017].
URL https://tools.ietf.org/html/rfc5694

[3] Chothia, T.; Cova, M.; Novakovic, C.; aj.: *The Unbearable Lightness of Monitoring: Direct Monitoring in BitTorrent*. 2012, [Online; navštíveno 18.12.2017].
URL https://www.semanticscholar.org/paper/The-Unbearable-Lightness-of-Monitoring%3A-Direct-in-Chothia-Cova/32d0092a886782efcde478ef7695d8d1ab1f21dc

[4] Chroboczek, J.: *BitTorrent DHT Extensions for IPv6*. Říjen 2009, [Online; navštíveno 10.2.2018].
URL http://www.bittorrent.org/beps/bep_0032.html

[5] Cohen, B.: *Incentives Build Robustness in BitTorrent*. Květen 2003, [Online; navštíveno 13.10.2017].
URL http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.1911

[6] Cohen, B.: *The BitTorrent Protocol Specification*. Únor 2017, [Online; navštíveno 14.10.2017].
URL http://www.bittorrent.org/beps/bep_0003.html

[7] Dave, N.; Huang, A.; Shen, Y.; aj.: *Content Distribution Using an Enhanced BitTorrent System*. 2004, [Online; navštíveno 14.10.2017].
URL https://www.semanticscholar.org/paper/Content-Distribution-Using-an-Enhanced-BitTorrent-Dave-Huang/6ebf75be7c4a08086f30aaacb28ce04bd7ecaeb1

[8] Foundation, P. S.: *sqlite3 - DB-API 2.0 interface for SQLite databases*. [Online; navštíveno 14.04.2018].
URL https://docs.python.org/2/library/sqlite3.html

[9] Kurose, J. F.; Ross, K. W.: *Computer Networking: A Top-Down Approach*. Pearson Education, 6 vydání, 2013, ISBN 978-0-273-76896-8.

[10] Loewenstern, A.; Norberg, A.: *DHT Protocol*. Květen 2017, [Online; navštíveno 14.10.2017].
URL http://bittorrent.org/beps/bep_0005.html

[11] Lutz, M.: *Learning Python*. O'Reilly Media, páté vydání, 2013, ISBN 978-1-449-35573-9.

[12] Piatek, M.; Kohno, T.; Krishnamurthy, A.: *Challenges and Directions for Monitoring P2P File Sharing Networks - or - Why My Printer Received a DMCA Takedown Notice*. 2008, [Online; navštíveno 17.12.2017].
URL http://static.usenix.org/legacy/events/hotsec08/tech/full_papers/piatek/piatek_html/

[13] Vaško, M.: *Monitorovanie peerov BitTorrent na základe informácií z distribuovanej hašovacej tabuľky*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, květen 2018, aktuálně v obhajobě.

[14] Wang, L.; Kangasharju, J.: *Measuring Large-Scale Distributed Systems: Case of BitTorrent Mainline DHT*. 2013, [Online; navštíveno 20.12.2017].
URL https://ieeexplore.ieee.org/document/6688697/

[15] Widmer, P.: *Torrent Recommendation System Based on Data Gathered from the Mainline DHT*. 2015, [Online; navštíveno 19.12.2017].
URL https://www.semanticscholar.org/paper/Torrent-Recommendation-System-Based-on-Data-from-Widmer-Decker/b4cff0f19e6cd2620342f66d5dbfe0ac272e93eb

[16] Wolchok, S.; Halderman, J. A.: *Crawling BitTorrent DHTs for Fun and Profit*. 2010, [Online; navštíveno 19.12.2017].
URL https://www.researchgate.net/publication/228363995_Crawling_BitTorrent_DHTs_for_fun_and_profit

[17] Zhang, C.; Dhungel, P.; Wu, D.; aj.: *Unraveling the BitTorrent Ecosystem*. 2010, [Online; navštíveno 16.12.2017].
URL https://ieeexplore.ieee.org/document/5482574/

Příloha A

Obsah DVD

Přiložené DVD obsahuje následující adresáře:

- Sources - Zdrojové soubory softwaru implementovaného v této práci, včetně přiloženého souboru README.md obsahující manual ke zdrojovým souborům.
- Data - Získaná data nasbírána během testování a vyhodnocení softwaru. Data byla zkomprimována do souboru DIP-data.zip z důvodu omezené kapacity DVD.
- Latex - Zdrojové soubory textové části zprávy.
- Text - Výsledná podoba textové zprávy.